LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

INSTITUT FÜR STATISTIK

Georg Pfundstein

Ensemble Methods for Plaid Bicluster Algorithm

Bachelor Thesis
Supervision: Dipl. Stat. Sebastian Kaiser – Prof. Dr. Friedrich Leisch
Department of Statistics – University of Munich

8th February 2010

Abstract

In recent years a lot of people came up with several biclustering algorithms, which perform simultaneous clustering on rows and columns of a data matrix. These methods are especially popular when analysing gene expression data, but can be applied in several other fields of two way data analysis as well. However, this thesis deals with the application of the plaid biclustering model to real gene expression data. Since the method has got some restrictions, that is mainly its sensitiveness against different parameter settings and repeated runs an ensemble method is introduced. The proposed ensemble method aims to overcome these limitations through repeatedly applying the plaid model with a different parametrisation. Out of these results the biclusters which were observed the most are said to be the best according to the introduced method. Based on these results conclusions on the parameter settings of the plaid model are drawn.

Keywords: Biclustering, Plaid Model, Ensemble Method, Jaccard Index, TCGA, Gene Expression Data, `biclust`

Acknowledgment

This work is mostly the result of an internship held in summer 2009 at the Bioinformatics Core Department of Memorial Sloan-Kettering Cancer Center (MSKCC) in New York City. For this reason, I would like to thank Nicholas Socci, Alex Lash and especially Raya Khanin from MSKCC for giving me the opportunity to work on this project as well as for their outstanding support during this period. I am equally grateful to Sebastian Kaiser and Friedrich Leisch from the University of Munich for their excellent supervision during my further work on this thesis. Furthermore, I send my thanks to Oliver Kühnle for proof-reading and the good times in New York, Glasgow and elsewhere. And last but not least I would like to thank my family for their assistance and support over all the years.

Contents

1. Introduction

The analysis of gene expression data (often also referred to as genomics data) has become a highly popular technique for studying the biological structure of human beings, or rather all kind of creatures. For example, one is able to discover groups of genes which share similar functions and characteristics under a specific condition. In other words, with the aid of genetic data scientists aim to achieve biologically relevant insights into the transcriptional network of different organism. Nowadays these data are routinely measured by so called DNA microarrays, which are able to monitor the activity of thousands of genes simultaneously. The gene expression data is then mostly given in a n by m data matrix, where the genes are arranged in rows and the conditions in columns. That means, element a_{ij} represents the expression level of gene i under condition j. Whereas the level of a certain gene can vary between sample groups (e.g diseased vs. normal) as well as different individuals.

Traditionally the analysis of those expression matrices can be done by applying standard clustering algorithms, such as K-means or hierarchical to the data. However, this approach has got a number of limitations. First, common cluster methods usually seek a disjoint cover of the set of elements, requiring that no gene or sample belongs to more than one cluster [Tanay et al., 2005]. In fact, genes can participate in more than one biological function and should therefore be included in several clusters. Another drawback is, that these algorithms classify genes on the basis of their expression under all experimental conditions, whereas cellular processes are generally affected only by a small subset of conditions [Ihmels et al., 2003]. Further, cluster algorithms can only be applied on either rows (genes) or columns (conditions), which leads to an very biased analysis of the genomic data. Yet another aspect is, that the clusters should not be exhaustive, that means the algorithm should allow some genes or samples not to be in a cluster at all.

To overcome all those limits bicluster (or two-way-cluster) algorithms have become more and more popular in recent years. A bicluster algorithm performs simultaneous clustering on rows and columns of an expression matrix or a data matrix in general. Hence, a bicluster is defined as a submatrix consisting of a subgroup of genes and a subgroup of conditions which are as different as possible to the rest and as similar as possible to each other. Thus, biclustering algorithms

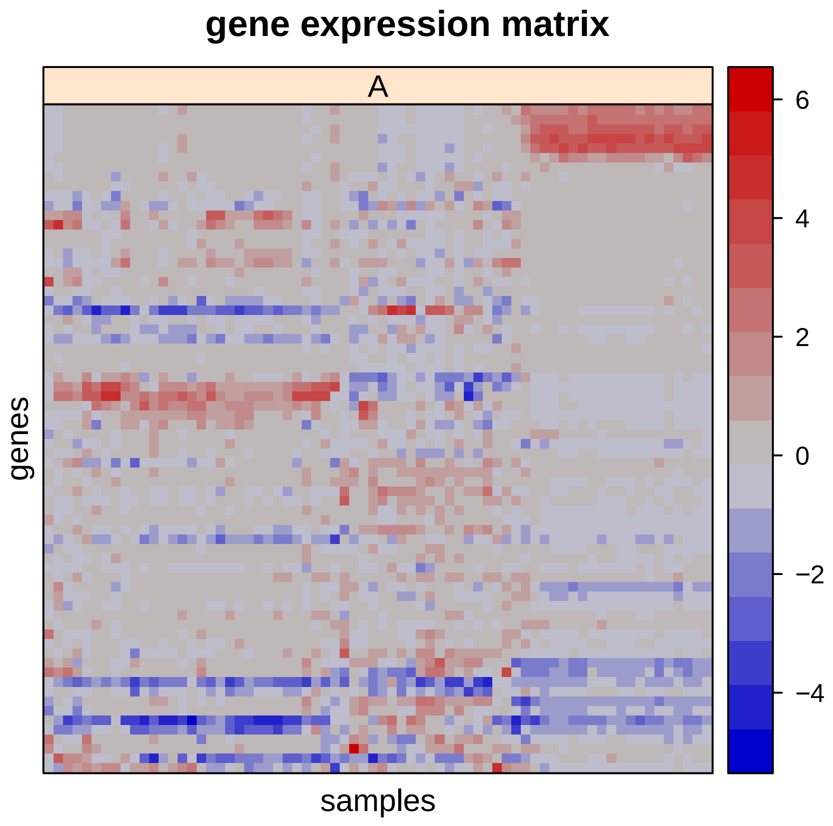

Figure 1.1.: A levelplot of a small section of the BicatYeast gene expression data set [Barkow et al., 2006], which shows the different expression levels of genes and samples.

are able to identify groups of genes that show similar expression pattern under a specific subset of the experimental conditions [Madeira and Oliveira, 2004]. However, most approaches which have been proposed over the last decade differ in their definition of a bicluster and therefore use quite different methods.

Biclustering is not only appropriate for the analysis of gene expression data, but also for several other research areas. E.g. in marketing biclusters can be used to identify a subgroup of consumers with similar preferences regarding a subset of products. This technique is often used to supply recommendation systems with proper information. Namely, to predict the consumers preferences and make suitable recommendations [Madeira and Oliveira, 2004]. Another application field is in the area of text mining, where one wants to derive information from a text, more precisely wants to obtain information about the structure of a text. In addition, one can think about more research areas where bicluster algorithm could be applied. However, this work focuses on the bioinformatics field, hence

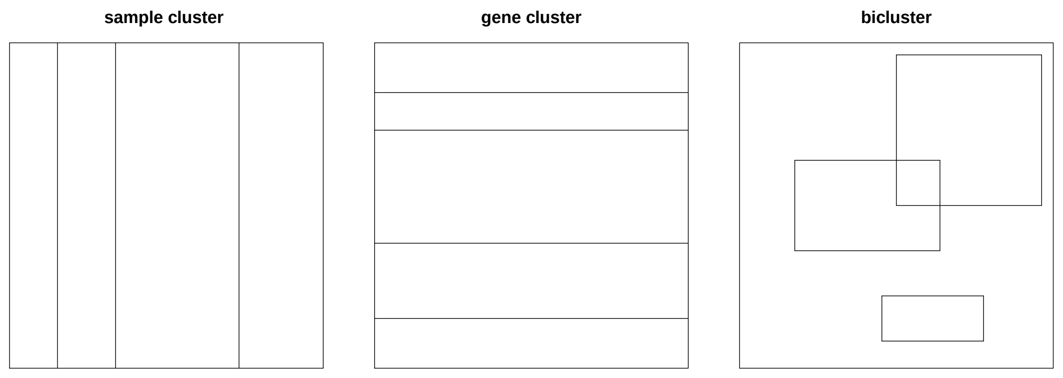

Figure 1.2.: Illustration of the difference between common cluster methods, which obtain cluster in only one direction (either rows respectively genes or columns respectively samples) and modern bicluster methods which are able to find subsets of rows and columns simultaneously.

on gene expression data.

In chapter 2 an overview of some of the most common bicluster algorithms is given. Based on some a priori requirements which are necessary when dealing with gene expression data the plaid algorithm is chosen to fit them best and is described in more detail. Chapter 3 introduces the R [R Development Core Team, 2009] package `biclust` [Kaiser and Leisch, 2008], which contains, amongst other things, a function to compute bicluster based on the plaid algorithm. Because it turned out that plaid is very sensitive against different parameters and independent runs two ensemble methods are proposed in chapter 4 which aim to overcome these limitations. In chapter 5 the introduced ensemble methods are applied on real gene expression data and conclusions based on their results are drawn. Finally in chapter 6 a discussion of this thesis and a prospect for further research in this context are presented.

2. Bicluster Algorithm

Below, different types and structures of biclusters are presented. Moreover an overview over some of the recently developed bicluster models and algorithms is given. Each of them is judged based on their capability to analyse gene expression data. Because this work especially deals with plaid bicluster model it is described in detail in section 2.3. Since there are plenty of different methods this survey, as a matter of course is not exhaustive.

2.1. Requirements on a Bicluster Algorithm

As already mentioned, most of the bicluster algorithms which have been proposed have got quite a different idea of what a bicluster actually is and how it should look like. And therefore they differ in their appropriate application area in which they could be applied. Due to the huge amount of different methods one needs to classify the algorithms on the basis of some criteria.

First of all, there are so called two-way clustering methods which use a combination of traditional row-wise and column-wise cluster results. Whereby this results are produced by applying standard clustering methods such as K-means or hierarchical on the rows and columns dimensions of the data matrix separately. In addition to that there are plenty algorithms which perform clustering on both dimensions simultaneously. The divide-and-conquer approach for example breaks the problem into multiple subproblems which are smaller in size but similar to the original problem. Other methods follow a greedy iterative search approach which are based on the idea of adding/removing rows or columns to a bicluster in order to maximise a local criterion. The exhaustive bicluster enumeration methods seek to identify the best biclusters by using an exhaustive enumeration of all possible biclusters in the data matrix. Certainly this methods can only be applied by assuming restrictions on the size of the biclusters. Moreover there are some model-based approaches whereas the distribution parameter have to be identified. Those methods assume a given statistical model and try to minimise a certain criterion by identifying the parameters that fit the data best.

Another criteria to distinguish bicluster methods is the type of bicluster the algorithm is able to find. The specialist literature gives four different types (examples are given in Figure 2.1) [Madeira and Oliveira, 2004]:

1. Biclusters with constant values.
2. Biclusters with constant values on rows or columns.
3. Biclusters with coherent values.
4. Biclusters with coherent evolutions.

In the simplest case the algorithm is able to find subsets of rows and columns with constant values, which indeed is not applicable on gene expression data. Slightly enhanced methods can identify bicluster with either constant values on the rows or constant values on the columns. Again this methods are not suitable for the use on genomics data. Other approaches look for coherent values on the columns or rows of the expression matrix. This means, each column or row can be calculated by simply adding or multiplying a constant to each other. A further type aims to find biclusters with coherent evolutions. In other words, the exact numeric value of the matrix elements does not matter. Instead the algorithm searches for subsets of columns and rows with coherent behaviors. Obviously, this only comes along with a loss of information as the matrix has to be discretized since the exact numeric values of the matrix does not matter.

constant values − overall

1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0

constant values − rows

1.0	1.0	1.0	1.0
2.0	2.0	2.0	2.0
3.0	3.0	3.0	3.0
4.0	4.0	4.0	4.0

constant values − columns

1.0	2.0	3.0	4.0
1.0	2.0	3.0	4.0
1.0	2.0	3.0	4.0
1.0	2.0	3.0	4.0

coherent values − additive

1.0	2.0	5.0	0.0
2.0	3.0	6.0	1.0
4.0	5.0	8.0	3.0
5.0	6.0	9.0	4.0

coherent values − multiplicative

1.0	2.0	0.5	1.5
2.0	4.0	1.0	3.0
4.0	8.0	2.0	6.0
3.0	6.0	1.5	4.5

coherent evolution − overall

S1	S1	S1	S1
S1	S1	S1	S1
S1	S1	S1	S1
S1	S1	S1	S1

coherent evolution − rows

S1	S1	S1	S1
S2	S2	S2	S2
S3	S3	S3	S3
S4	S4	S4	S4

coherent evolution − columns

S1	S2	S3	S4
S1	S2	S3	S4
S1	S2	S3	S4
S1	S2	S3	S4

Figure 2.1.: Examples of the different types of bicluster. The figure is based on Madeira and Oliveira [2004].

Additionally one can differ the methods on the basis of the bicluster structure a algorithm is able to identify. Here, the structure describes how the observed biclusters can potentially be arranged, whereas the following structures are supported by the different models (again, examples are given in Figure 2.2) [Madeira and Oliveira, 2004]:

1. Single bicluster.
2. Exclusive row and column biclusters.
3. Exclusive-rows or exclusive-columns biclusters.
4. Non-overlapping non-exclusive biclusters.
5. Arbitrarily positioned overlapping biclusters.

Structure type 1 is self-explanatory as the method is just able to find one single bicluster at a time. In type 2 every bicluster consists of an exclusive subset of rows and columns. Although this can be the first approach to extract relevant knowledge from gene expression data, it has long been recognized that such an structure will seldom exist in real data [Madeira and Oliveira, 2004]. As mentioned in the last chapter genes are very likely to participate in more than just one biological function and therefore should be included in more than one bicluster. The same of course holds for the conditions. That is, where it comes to biclusters of type 3. Improvements of this structures are the non-overlapping and non-exclusive biclusters, which exhaustively splits the matrix in biclusters which overlap in just one dimension. But still, this methods have some restrictions, such as every row and column in the data has to belong to at least one bicluster. However, in the context of gene expression it is very likely that some rows or columns do not belong to any bicluster at all. Thus, there are algorithms which are able to observe arbitrarily positioned overlapping biclusters. This sophisticated methods allow bicluster to be non-exclusive, non-exhaustive and overlapping in both dimensions.

When it comes to the analysis of gene expression data one - in the ideal case - wants to be able to observe overlapping, non-exclusive and non-exhaustive biclusters with coherent values (either additive or multiplicative). The reasons: genes participate in more than one biological function and some conditions share certain gene expression alterations (non-exclusive and overlapping), some genes are not relevant at all (non-exhaustive) in an certain experiment and at last, gene expression level varies between different genes and samples and one does not want to lose information (coherent values), have already been mentioned above.

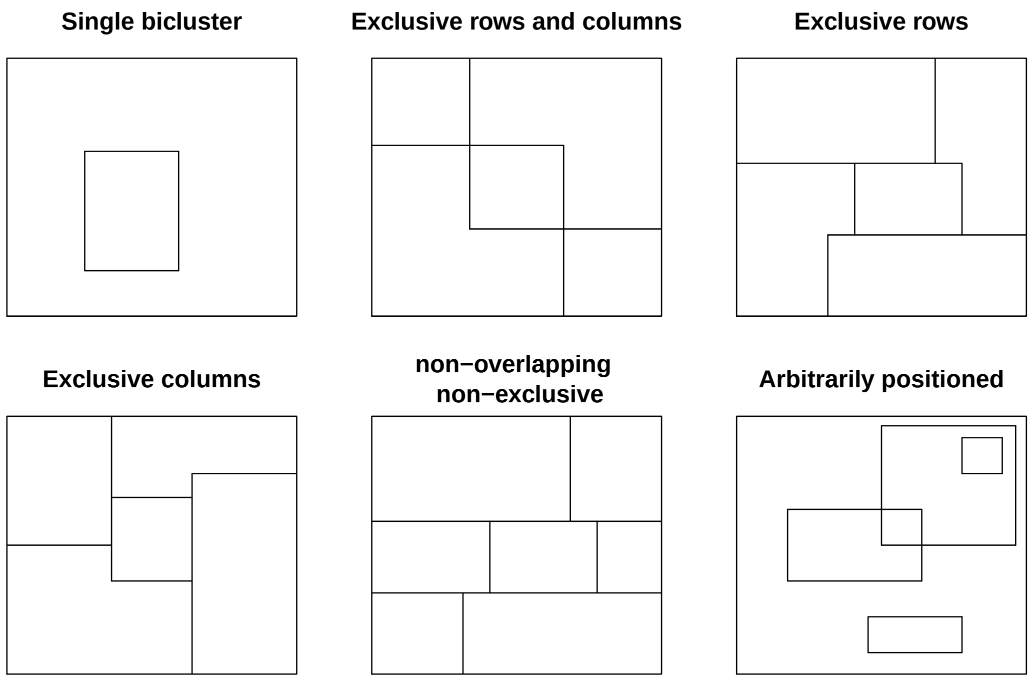

Figure 2.2.: Examples of the different bicluster structures. Note, that rows and columns of a single bicluster typically are not together in a block, they rather have to be reordered in order to be displayed in a "nice" block structure. Again the figure is based on Madeira and Oliveira [2004].

2.2. Overview over some Methods

In this section some of the most important and sophisticated algorithms for bicluster analysis are described in short and rated based on the requirements mentioned in the last section. In addition to that a table summarizing more methods is given in appendix B.

One of the first who introduced a basic idea of biclustering to the research community was Hartigan [1972]. His approach is an example of a divide-and-conquer method, which aims to minimize the overall variance within a cluster (sum of squares) by splitting a given partition into two cluster. The splitting which maximizes the sum of squares reduction is chosen. The splitting of the matrix is stopped when a further splitting reduces the sum of squares less than expected by chance. However, this method just can handle constant bicluster and only allows overlapping in one dimension. In addition to that the observed biclusters cover the data matrix completely.

Pioniers in applying bicluster algorithms to gene expression data were Cheng and Church [2000]. The basic idea behind their greedy iterative search approach called the δ-method is, that there are specific row, column and submatrix effects within a bicluster. Thus, each bicluster has got a residual score

$$H(I,J) = \frac{1}{|I||J|} \sum_{i \in I, j \in J} (a_{ij} - a_{iJ} - a_{Ij} + a_{IJ})^2, \tag{2.1}$$

where a_{iJ} is the mean of row i (row effect), a_{Ij} is the mean of column j (column effect) and a_{IJ} is the overall mean of the bicluster (submatrix effect). If the score of a certain submatrix A_{IJ} is below a threshold δ the submatrix is called a bicluster. The algorithm starts from the entire matrix A and deletes rows and columns with the highest score as long as $H(I,J) > \delta$. In a second step rows and columns with the lowest score are being added as long as $H(I,J) < \delta$. The described process is repeated until an appropriate number of biclusters is found. Cheng and Church suggested to mask previously detected bicluster by random numbers, which makes it unlikely to identify overlapping biclusters.

An example for a two-way clustering method is the Coupled Two-Way Clustering (CTWC) described by Getz et al. [2000]. As noted in the last section two-way cluster methods apply standard clustering techniques on the row and column dimension of a data matrix. The biclusters are then a combination of both results. First, the data matrix is clustered into subsets of genes and subsets of columns separately. After this, CTWC applies one dimensional clustering methods twice (on the row and column dimension) of each submatrix defined by all possible combinations of the row and column subsets obtained in the first step.

Yet another biclustering method is the Iterative Signature Algorithm (ISA) introduced by Bergmann et al. [2003]. This algorithm assumes that in a bicluster the average gene expression over all genes for each sample and the average gene expression over all samples for each gene should be unusually high or low. In order to obtain such biclusters the algorithm generates two normalized copies of the gene expression matrix, one with normalized rows A^G and one with normalized columns A^C. A bicluster $A_{I'J'}$ is then defined by

$$I' = \{i \in I : |a^C_{I'j}| > T_C \sigma_C\}, \quad J' = \{j \in J : |a^G_{iJ'}| > T_G \sigma_G\} \tag{2.2}$$

where $a^C_{I'j}$ is the mean expression level of genes from I' in the condition j and by $a^G_{iJ'}$ the mean expression level of gene i in conditions from J'. T_G and T_C are the threshold parameters and σ_C, σ_G are the standard deviations of the corresponding means $a^C_{I'j}$ and $a^G_{iJ'}$. This implies that the z-score of each gene, measured with respect to the bicluster conditions should exceed a certain threshold T_G. Similarly, the z-score of each condition, measured with respect to the bicluster genes should exceed the threshold T_C. In other words, the average expression of genes/conditions over the conditions/genes should be significantly far apart from its expacted value on random matrices. The algorithm itself starts with a random set of genes I' and then repeatedly applies the equation (2.2) on the samples and genes alternately until it converges to a fixed point. ISA meets all of the necessary requirements which have been discussed in section 2.1.

The bicluster algorithm proposed by Kluger et al. [2003] uses a totally different approach. Their Spectral biclustering method suggests to use singular value decomposition (in short, SVD a technique from linear algebra which factorizes a matrix A in the form $A = U\Sigma V$, where U and V are orthonormal matrices and Σ is a diagonal matrix) to get eigenvalues and eigenvectors. After normalizing the matrix A one is able to observe the biclusters by solving the coupled eigenvalue problem $A^T Ax = \lambda^2 x$ and $AA^T y = \lambda^2 y$. The column clusters are then found by sorting the eigenvectors x to a step-like structure. Row clusters are found similar by sorting the eigenvectors y. The biclusters are built beginning from the largest eigenvalue. However, this model assumes a checkerboard-like structure which implies that no real overlapping of the biclusters is possible.

An example for a method which searches for coherent evolution within a bicluster is the algorithm of Murali and Kasif [2003] called xMotifs. As the coherent evolution structure implies, the data matrix has to be discretized first. After this the method looks for groups of columns in which a subset of the rows is in the same state. Starting by n randomly selected columns the algorithm chooses a subset of all samples for each selected column uniformly at random. For each of this sets of columns the algorithm then collects rows with equal state across this subset of columns. In a further step columns are collected where this subset of rows have the same state. The obtained set of rows and columns is then called a bicluster if it contains more than a α fraction of all samples.

As already mentioned at the beginning of this section, an summarizing table regarding the requirements on biclusters for this and several more methods is given in appendix B

2.3. Plaid Method

The plaid model, originally proposed by Lazzeroni and Owen [2000] is a very flexible and powerful model-based bicluster method which meets all the necessary requirements. Each element of the data matrix can be viewed as a sum of terms, whereas the terms are called layers and correspond to biclusters in our case.

2.3.1. Model

The idea behind the plaid model is, that a data matrix can be described as a linear function of layers. That means, the expression level of a matrix is the sum of K biclusters and a uniform background noise term. To specify, the matrix entries can be written as

$$a_{ij} = \sum_{k=0}^{K} \theta_{ijk}\rho_{ik}\kappa_{jk} \tag{2.3}$$

where $\rho_{ik} \in \{0,1\}$ and $\kappa_{jk} \in \{0,1\}$ are indicator variables indicating whether a gene i or condition j belongs to the k-th bicluster or not. Note, by setting some constraints on ρ and κ one is able to allow or prohibit overlapping or exhaustiveness. For example, by allowing $\sum_k \rho_{ik} = 0$ and $\sum_k \kappa_{jk} = 0$ for some i and j there could be some genes/conditions which do not belong to any bicluster at all. Also θ_{ijk}, which specifies the contribution of each element to the matrix can be defined very flexible according to identify different types of biclusters. In this case θ_{ijk} is defined as $\mu_k + \alpha_{ik} + \beta_{jk}$, where μ_k is the background noise term of bicluster k and α_{ik} and β_{jk} describes the row and column specific effects in a bicluster k. Which is quite similar to the method introduced by Cheng and Church [2000]. Thus, the model may then also be written as

$$a_{ij} = \mu_0 + \sum_{k=1}^{K} (\mu_k + \alpha_{ik} + \beta_{jk})\rho_{ik}\kappa_{jk} \tag{2.4}$$

where μ_0 is the background noise term of the whole data matrix.

2.3.2. Estimation

The model described above can now be formulated as the minimization problem

$$\text{argmin} \left[\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} (a_{ij} - \theta_{ij0} - \sum_{k=1}^{K} \theta_{ijk}\rho_{ik}\kappa_{jk})^2 \right]. \tag{2.5}$$

Suppose we already found $K-1$ layers and want to observe the K-th layer. The minimization problem can then be rewritten as

$$\text{argmin} \left[Q^{(K)} = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} (Z_{ij}^{(K-1)} - \theta_{ijk}\rho_{ik}\kappa_{jk})^2 \right], \tag{2.6}$$

with

$$Z_{ij}^{(K-1)} = a_{ij} - \theta_{ij0} - \sum_{k=1}^{K-1} \theta_{ijk}\rho_{ik}\kappa_{jk}. \tag{2.7}$$

To solve this equation an iterative approach is proposed which updates θ, ρ and κ values in each cycle alternately. Given a priori defined number of iterations S and initial parameters $\rho^{(0)}$ and $\kappa^{(0)}$ the algorithm proceeds as follows: At each iteration s, optimal values for the $\theta^{(s)}$ parameters are computed given fixed $\rho^{(s-1)}$ and $\kappa^{(s-1)}$ values, afterwards the optimal value for $\rho^{(s)}$ is computed given new $\theta^{(s)}$ and old $\kappa^{(s-1)}$ values, and finally $\kappa^{(s)}$ is updated by using $\theta^{(s)}$ and $\rho^{(s-1)}$ values. Whereas the following estimations for the iteratively update of the θ parameters, given ρ and κ values can be obtained by using straightforward Lagrange multipliers:

$$\mu_K = \frac{\sum_i \sum_j \rho_{iK}\kappa_{jK} Z_{ij}^{(K-1)}}{(\sum_i \rho_{iK}^2)(\sum_j \kappa_{jK}^2)} \tag{2.8}$$

$$\alpha_{iK} = \frac{\sum_j (Z_{ij}^{(K-1)} - \mu_K \rho_{iK}\kappa_{jK})\kappa_{jK}}{\rho_{iK} \sum_{jK} \kappa_{jK}^2} \tag{2.9}$$

$$\beta_{jK} = \frac{\sum_i (Z_{ij}^{(K-1)} - \mu_K \rho_{iK}\kappa_{jK})\rho_{iK}}{\kappa_{jK} \sum_{iK} \rho_{iK}^2} \tag{2.10}$$

The optimal parameter values for the row and column membership indicators ρ and κ, which minimises the equation (2.6) are given by:

$$\rho_{iK} = \frac{\sum_j \theta_{ijK}\kappa_{jK} Z_{ij}^{(K-1)}}{\sum_j \theta_{ijK}^2 \kappa_{jK}^2} \tag{2.11}$$

$$\kappa_{jK} = \frac{\sum_i \theta_{ijK}\rho_{iK} Z_{ij}^{(K-1)}}{\sum_i \theta_{ijK}^2 \rho_{iK}^2} \tag{2.12}$$

After repeating the above described cycle predefined S times the algorithm accepts a bicluster if its importance is significantly larger than one obtained in noise. Whereas the importance of a layer k is defined by $\sigma_k^2 = \sum_{i=1}^{n} \sum_{j=1}^{m} \rho_{ik}\kappa_{jk}\theta_{ijk}^2$. The procedure described above is repeated until a bicluster is rejected due to his small importance. A short pseudo code of this algorithm is outlined in figure 2.3.

As mentioned above the algorithm requires initial values of ρ and κ. Reasonable values can be obtained by finding singular vectors u and v and a real value λ such that λuv^T is the closest rank one approximation of Z [Tanay et al., 2005]. However, to obtain such singular vectors one again needs release values for each

ρ and κ. For more details see the original paper by Lazzeroni and Owen [2000].

```
Set number of cycles $S$
Set $K = 0$
Compute new bicluster:
    $K = K + 1$
    Set $s = 1$
    Compute initial values of $\rho^{(0)}$ and $\kappa^{(0)}$.
    **While** $(s \leq S)$ do:
        Compute $\mu_K^{(s)}$, $\alpha_K^{(s)}$ and $\beta_K^{(s)}$ using equations (2.8)-(2.10).
        Compute $\kappa_K^{(s)}$ using equation (2.11).
        Compute $\rho_K^{(s)}$ using equation (2.12).
        Set $s = s + 1$
    Compute the importance $\sigma_K^2$.
    **If** the importance is larger than by random save bicluster and repeat.
    **Else** exit.
Return biclusters.
```

Figure 2.3.: A pseudo code of the Plaid algorithm following Tanay et al. [2005].

3. R package `biclust`

A slightly adapted version (ordinary least square estimation is replaced by an binary least square algorithm) of the plaid algorithm was implemented in the statistical computing environment R [R Development Core Team, 2009] by Kaiser and Leisch [2008]. More precisely, they developed a general framework for biclustering within R, which besides the plaid model also contains several other bicluster algorithms. For example: Spectral, xMotifs, or the δ-method proposed by Cheng and Church. However, their package `biclust` not only provides methods to compute biclusters, but also a few other useful functions for the preprocessing (binarization, discretization and normalization) of the data matrix and the validation and visualization of bicluster results. The package is freely available in version 0.91 on R-Forge [Theußl and Zeileis, 2009].

The function to compute biclusters based on the plaid model, which was used in this work can be called by the following command (all arguments are set to their default value):

```
biclust(x, method=BCPlaid(), cluster="b", fit.model = y ~ m + a + b,
        background = TRUE, row.release = 0.7, col.release = 0.7,
        shuffle = 3, back.fit = 0, max.layers = 20, iter.startup = 5,
        iter.layer = 10, verbose = TRUE)
```

Table 3.1 gives a short description of all the arguments which are included in the function above. After running the function on a data matrix it returns an object of the class `Biclust` which contains, among other things, the obtained bicluster. An overview over all returned slots is given in table 3.2.

3.1. Influence of the Parameters

An earlier thesis written by Felix Herzog dealt with the issue of applying the plaid model to simulated data. The aim was to figure out how the results differ when changes in the parameters are made and which parameter settings are the best,

Parameter	Details
`x`	The data matrix in which biclusters should be found.
`method`	Choice of the method. The argument `BCPlaid()` performs the plaid algorithm.
`cluster`	`"r"` clusters rows, `"c"` columns and `"b"` (default) both.
`fit.model`	Linear model to be fit. The formular is similar to the θ parameter in the model description. `m` is the overall bicluster constant μ, `a` the row constant α and `b` the column constant β.
`background`	If `TRUE` the function will allow an overall background layer in the data matrix.
`row.release`	Threshold to prune rows in the layers. Scalar in `[0,1]` with recommended interval `[0.5,0.7]`
`col.release`	Same as `row.release`, but for columns.
`shuffle`	Number of random layers to compute the significance of an observed layer. Default is set to 3.
`back.fit`	Additional iterations to refine the fitting, after a layer was found (default set to 0).
`max.layers`	Maximum number of bicluster to be found.
`iter.startup`	Number of iterations to find starting values.
`iter.layer`	Number of iterations to find a bicluster.
`verbose`	If `TRUE` extra information on progress is printed.

Table 3.1.: Overview over the parameters of the `biclust` function.

Slot	Details
`Parameters`	Contains a list of the input parameters.
`RowxNumber`	Logical Matrix which gives the row bicluster membership. `TRUE` in $[i, j]$ if row i is in bicluster j.
`NumberxCol`	Same as `RowxNumber`, but for columns. `TRUE` in $[i, j]$ if column j is in bicluster i.
`Number`	Number of observed bicluster.
`info`	Additional information on the bicluster. For example Sum of Squares (SS) and Mean Sum of Squares (MS)

Table 3.2.: Overview over the return values of the class `Biclust`.

given specific data sets. As a result, it can be stated that there is no such thing like an optimal parameter setting for all possible types of data. Furthermore it is still not totally clear which parameter settings to use given a specific data set. In fact, the results are highly dependent on the selected `row.release` and `col.release` values. In contrast, changes in the parameters `shuffle`, `back.fit`, `iter.startup` and `iter.layer` do not seem to have a huge effect on the results. Another important insight is the fact that even with identical parameter settings the algorithm is likely to observe different biclusters in different runs. However, given appropriate parameter settings and repeated runs the plaid model is powerful enough to find the real biclusters when applied on simulated data.

4. Ensemble Methods

As mentioned before, it is not totally clear which parameter settings are the best for a given data set. Furthermore, it turned out that the plaid method is very sensitive, also for exactly the same settings (i.e. release levels) the results diver in an not irrelevant way. All in all these of course are not the ideal conditions for working with real data sets. However, besides it also turned out that given appropriate parameter settings and repeated runs the plaid model in fact is able to find the real biclusters. For these reasons one has to think about a method which overcomes the disadvantages of unknown parameter settings and differing results within a particular parametrisation. After all, the plaid model should be able to obtain the real biclusters without carrying about the settings and different runs.

Given the fact, that the algorithm indeed is able to observe the real biclusters one may assume that the real biclusters will be obtained more often than random biclusters when the method is applied several times. In contrast, biclusters which were found at random may occur just once or a very few times, depending on the number of cycles. These assumptions brings us to the basic idea behind the proposed ensemble method, which works like follows: After repeatedly applying the `biclust` function with different parameter settings several times the obtained results are merged to an overall result. That are matrices R, indicating the row bicluster membership of all observed biclusters and C indicating the column bicluster membership (after transposing the original column-membership matrices). Since only the `row.-` and `col.release` levels seem to have a relevant influence on the bicluster results only these two values should vary (within its recommended interval of $[0.5, 0.7]$) between the different runs, the rest of the parameters can be set to its default values. Whereas each parametrisation should be applied multiple times (e.g. 100 times) on the data. In a next step the assembled results are compared with the help of some similarity measure. After choosing a threshold for the similarity one is able to rate the biclusters according to their frequency. Following the biclusters which do not share any or a predefined number of similar biclusters are cut off, because they are not likely to be the real ones. In addition, due to the distribution of the frequency values one also can decide to cut off a certain quantile of the biclusters which occurred the least. For example the lower 75%-quantile, so that only a quarter (the 25% of biclusters which occurred the most) are kept and said to be the best, in other words real biclusters according to the proposed ensemble method. Of course, this value can be modified and one can cut off a larger or smaller quantile of the biclusters. Following, one has to decide which bicluster per similar biclusters to keep since one does not want to keep the whole amount of similar biclusters, but only one. An obvious way is to pursue one of the following approaches: Keep the biggest bicluster, keep the smallest bicluster or an more sophisticated approach would include rows respectively columns which are represented in more than a certain percentage (for example 50%) of the similar biclusters. Here, however, only a method which keeps the first obtained bicluster per similar biclusters was implemented.

In order to get the number of similar biclusters one has to think about a method to compare the whole amount of obtained biclusters. There are two obvious ways to face this problem, one of which would be, to compare the row and column vectors of two different biclusters separately. The other way would be to see a bicluster as a subset and therefore to compare the subsets instead of comparing each bicluster dimension separately as suggested before. In the following subsections two approaches are presented and described. In addition, the R-Code of the introduced methods is given in Appendix D as well as on the attached CD.

4.1. Correlation Approach

The first method which is going to be proposed here is based on the correlation between the biclusters. More precisely on the separate correlations between each pairwise row- and column-membership vector combination of the biclusters.

In other words two correlation matrices R^{Cor} and C^{Cor}, where the elements r_{ij}^{Cor} and c_{ij}^{Cor} are representing the correlation between the vectors $R_{\cdot j}$ and $R_{\cdot i}$ and the vectors $C_{\cdot j}$ and $C_{\cdot i}$ respectively are computed. Since the vectors are binary the correlation had to be calculated with the Φ-coefficient. However, the Φ-coefficient is in fact equal to the pearson correlation, when applied on two binary variables. As two biclusters should not only be marked as similar with a match of a hundred percent (correlation of 1) but also with a small variation in genes or samples one has to think about appropriate values at which correlation the biclusters should be marked as similar. An adequate divergence in each dimension is for example 5%, however, one can think of other values as well. Since correlation can not be equated with percentage divergence one has to figure out which correlation threshold leads to the desired allowed tolerance. In most cases the dimensions of the data matrix are extremely different, therefore it is suggested to choose threshold values t_R and t_C for each dimension separately. Hence two row, respectively column vectors i and j are marked as similar when:

$$r_{ij}^{Cor} \geq t_R \tag{4.1}$$

$$c_{ij}^{Cor} \geq t_C \tag{4.2}$$

Finally, biclusters are set to be similar if both, the row- and column-membership vectors were marked as similar. Thus, one is able to get the number of similar biclusters for each obtained bicluster. Following, one can proceed as described in the section above. That is, cutting of biclusters which occurred just a few times and deleting similar biclusters in order to get the best ones.

4.2. Jaccard Index Approach

Another method, which follows the approach of looking at a bicluster as a subset is based on the Jaccard Index [Jaccard, 1901], which is also implemented in the `biclust` package. However, a slightly adapted version of the proposed code was used in this thesis. The Jaccard Index measures the similarity of sample sets and is defined as the amount of similar elements divided by the size of the union of two sample sets. Thus, given two biclusters A and B the Jaccard Index is defined by

$$JAC(A,B) = \frac{|A \cap B|}{|A \cup B|}. \tag{4.3}$$

Hence, two biclusters with one hundred percent similarity have got a jaccard index of 1 and two biclusters with no similar elements have got a jaccard index of 0. After computing the Jaccard Index for all observed bicluster combinations the elements jac_{ij} of the matrix JAC are representing the Jaccard Index between bicluster i and j and can be compared with a threshold value t_{JAC}. In other words, two bicluster i and j are marked as similar when:

$$jac_{ij} \geq t_{JAC} \tag{4.4}$$

Again, one has to think about a appropriate threshold value. Since the correlation approach described in the last section suggests a divergence in each dimension of about 5%, which is in fact equal to a similarity in elements of around $0.95*0.95 = 0.9025 \approx 90\%$ the threshold for this approach could be set to 0.9. However, other threshold values are worth thinking about. Finally, again one is able to get the best biclusters by following the steps proposed in the main section of this chapter.

4.3. Score

In many situations one might want to know which of the resulting biclusters (after applying the ensemble method) is the best or whether there are differences in the quality of the biclusters. In other words, the order of the goodness of the bicluster could be of interest. In order to rank the resulting biclusters according to their goodness a simple score could be set up for each bicluster. As mentioned at the beginning of this chapter the real biclusters will be obtained more often than random ones. According to that, a bicluster which was found more often is more likely to be a real bicluster, thus more likely to be a better one. Hence the score which is proposed here represents how often a specific bicluster was found. In order to make biclusters obtained with different runs (i.e. different amount of in total computed models) comparable the score has to be standardised. An obvious standardisation method is to divide by the number of total runs. Hence, the score of a resulting bicluster i is defined as

$$S_i = \frac{x}{yz}, \tag{4.5}$$

where x is the number of biclusters which were marked as similar to bicluster i, y is the number of different parametrisations used in the ensemble method and z is the number of runs per parametrisation. Since each bicluster could only be found once per run the score is defined in the interval $[0, 1]$.

5. Application on real Gene Expression Data

In this chapter the plaid algorithm alone as well as in combination with the introduced ensemble methods is applied on real gene expression data. Furthermore, the results obtained with both methods are compared and conclusions on the right release levels are drawn. Again, the complete R-Code of the following analyses is given in Appendix D as well as on the attached CD.

5.1. TCGA Data

The TCGA gene expression data contains the RNA expression for cancer samples of The Cancer Genome Atlas[1] project and is so far one of the biggest expression datasets in the world. "The Cancer Genome Atlas aims to catalogue and discover major cancer causing genome alterations in large cohorts of human tumours through integrated multi-dimensional analysis."[McLendon et al., 2008] To provide the data to the research community, all TCGA data are available for the public at the Data Coordinating Center (DCC)[2].

The experiments were performed by the Broad Institute of MIT and Harvard using the Affymetrix (a manufacturer of DNA microarrays) microarrays in seven different institutes which are spread all over the United States. However, the TCGA data set this thesis deals with was already preprocessed by Nicholas D. Socci from the Computational Group of Memorial Sloan-Kettering Cancer Center (MSKCC - http://cbio.mskcc.org) in New York City. The Sloan-Kettering Cancer Center is one of the most important cancer researching and treatment institution worldwide.

The data consists of the RNA expression level of $n = 12042$ different human genes $G = (G_1, G_2, ..., G_n)$ (genes were given in symbols names) and $m = 202$

[1] http://cancergenome.nih.gov/
[2] http://tcga-data.nci.nih.gov

samples. Whereas the vector $S = (S_1, S_2, ..., S_m)$ represents the different types of brain cancer (type C with 50 samples, M with 63 samples, N with 33 samples and P with 56 samples). The expression data was transformed with the natural logarithm, which is a common procedure when dealing with RNA data. Hence the gene expression matrix looks like the following:

$$\begin{array}{c} \\ G_1 \\ G_2 \\ \vdots \\ G_n \end{array} \begin{array}{c} \begin{array}{cccc} S_1 & S_2 & \dots & S_m \end{array} \\ \begin{pmatrix} g_{11} & g_{12} & \cdots & g_{1m} \\ g_{21} & g_{22} & \cdots & g_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ g_{n1} & g_{n2} & \cdots & g_{nm} \end{pmatrix} \end{array}$$

5.2. Problems with real Data

For testing purposes the plaid algorithm was applied with three different `row.`- and `col.release` settings (0.5, 0.6 and 0.7) on the TCGA data set. Whereas the model was calculated twice for each release level in order to see whether there in fact appear different results within a similar parametrisation. The rest of the parameter were set to the following values:

Parameter	value
`cluster`	`"b"`
`fit.model`	`y ∼ m + a + b`
`background`	`TRUE`
`shuffle`	`3`
`back.fit`	`1`
`max.layers`	`100`
`iter.startup`	`3`
`iter.layer`	`5`

Table 5.1.: Used parameter settings for testing purposes.

To summarize, it can be stated that not only the number of observed biclusters is quite different within a release level, but also the biclusters seem to be pretty different as there sizes are unequal. For example, by applying the model with a `release` level of `0.6` the algorithm obtained four bicluster in the first run and two in the second. Corresponding heatmaps of the observed biclusters can be found in figure 5.1 and 5.2. Indeed there were huge differences in the obtained biclusters and obviously there were no bicluster which were found in both runs. That means, by looking at the observed bicluster results the conclusions outlined in section 3.1 can be validated. This certainly is a very unsatisfying result for working with gene expression data and again strengthen the need for a method which overcomes these limitations.

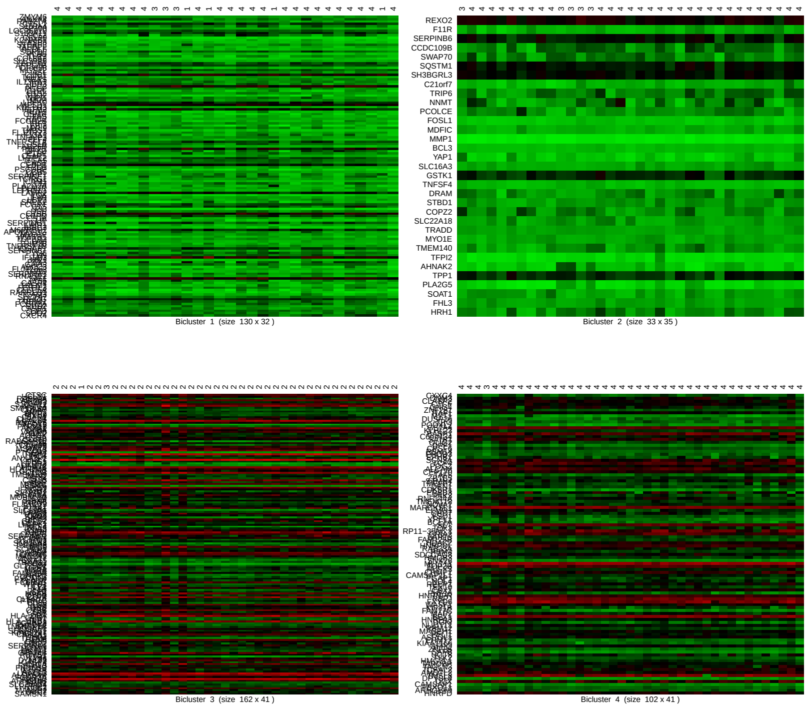

Figure 5.1.: Bicluster obtained with the first run.

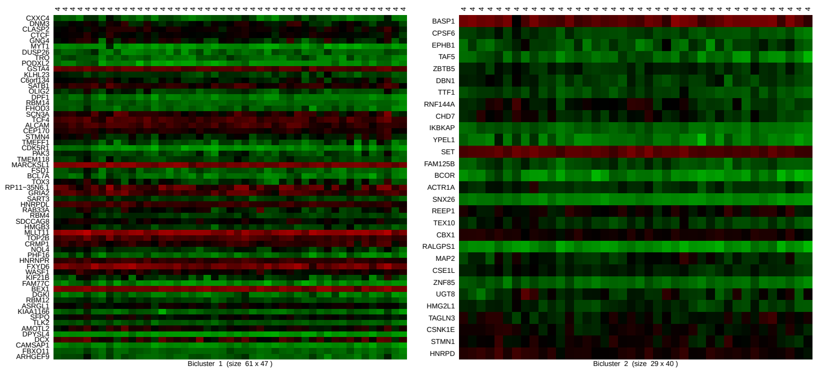

Figure 5.2.: Bicluster obtained with the second run.

5.3. Ensemble Methods

In order to apply the proposed ensemble methods in practice, the plaid algorithm was applied for `row.-` and `col.release` levels varying from 0.51 up to 0.71 in steps of 0.02 and the settings given in table 5.2. Whereas both release levels were set equal in each run. Each model was computed 100 independent times on a cluster computer setup with 50 simultaneous processes, which makes in total 1100 different results of the class `Biclust`. As mentioned at the beginning of this chapter the R-Code can be found in Appendix D. Altogether there were $T = 6567$ biclusters found. Afterwards the different results, which are presented through the slots `RowxNumber` and `NumberxCol` were combined to an overall results matrix. Thus, as mentioned in chapter 4 one got two new matrices indicating the row and column bicluster membership of all 6567 results at the same time. Hence, the new matrix R indicating row membership is of the dimension 12042 by 6567 (n by T) and the new column membership matrix C is of the dimension 202 by 6567 (m by T).

Parameter	value
`cluster`	`"b"`
`fit.model`	`y ~ m + a + b`
`background`	`TRUE`
`shuffle`	`3`
`back.fit`	`0`
`max.layers`	`100`
`iter.startup`	`15`
`iter.layer`	`30`

Table 5.2.: Used parameter settings for the ensemble method.

In the following subsections the two approaches described in chapter 4 are applied on the 6567 biclusters mentioned above. Besides, their results are compared and conclusions on the release levels are drawn.

5.3.1. Correlation Approach

First of all, (as described in section 4.1) one has to choose threshold values for the row- and column-correlation matrices. Due to the extremely different row and column sizes of the expression matrix the threshold should be chosen different in each dimension. Examples of different threshold values and their corresponding allowed tolerance in rows respectively columns is shown in table 5.3. Obviously, since small thresholds allow to much variation in big biclusters and large thresholds allow too less variation in small biclusters there is no such thing like a perfect threshold for all situations. The ideal case would be to have threshold values depending on the size of the expression matrix as well as the size of the bicluster. But this issue needs further studies, for the moment one needs to strike a balance between those two extremes. By looking at table 5.3 the row threshold was set to 0.95 and the column threshold to 0.9 since those two values allow the proposed divergence in each dimension of about 5%. That means, row vectors with a correlation greater than 0.95 and column vectors with a correlation greater than 0.9 are marked as similar. Thus, one is able to get the number of similar biclusters for each of the 6567 obtained biclusters.

	gene threshold			
size	0.8	0.85	0.9	0.95
25	0.2	0.16	0.08	0.04
50	0.2	0.14	0.1	0.04
100	0.2	0.14	0.1	0.05
150	0.2	0.14	0.1	0.05
200	0.2	0.15	0.1	0.05
400	0.2	0.15	0.1	0.05

	sample threshold			
size	0.8	0.85	0.9	0.95
25	0.16	0.12	0.07	0.04
30	0.16	0.14	0.06	0.03
40	0.15	0.13	0.06	0.02
50	0.14	0.12	0.07	0.03
80	0.13	0.09	0.05	0.02
100	0.1	0.08	0.04	0.03

Table 5.3.: The table shows the allowed approximate percentage tolerance in genes, respectively samples depending on the correlation thresholds and biclusters sizes. Due to the different row (*length* = 12042) and column (*length* = 202) sizes of the expression matrix the biclusters sizes are also different in each dimension. Based on this values the row threshold was set to 0.95 and the column threshold to 0.9, which allows an variation in each dimension of around 5%.

The numbers of similar biclusters in this data set is shown in figure 5.3. In fact, the majority of the biclusters was just found once or a few times. However, there are also a lot of biclusters which were found several times, in some cases up to 124 times. Due to this distribution only the upper 25%-quantile of biclusters (the ones which occurred the most) were kept. After applying the whole procedure described in chapter 4 there were 58 out of the 6567 biclusters left and said to be the real underlying biclusters in the data. The size of the biclusters varied in the gene dimension between 2 and 450 ($median = 139$; $mean = 153.6$) and in the sample dimension from 28 to 52 ($median = 41$; $mean = 39.3$). The biclusters were found between 25 and 94 times ($median = 41$; $mean = 35.9$). In total 8909 genes were included in any bicluster, thereof 1026 of them were unique.

Which makes an unique-gene-rate (# unique genes /# total genes) of 11.52%. There were 15 different genes which were included in 29 different biclusters. The distribution of the bicluster scores is shown in figure 5.5. The positive skewed distribution again indicates, that there are some biclusters which seem to be better than the rest, since they have a higher score, hence were found more often.

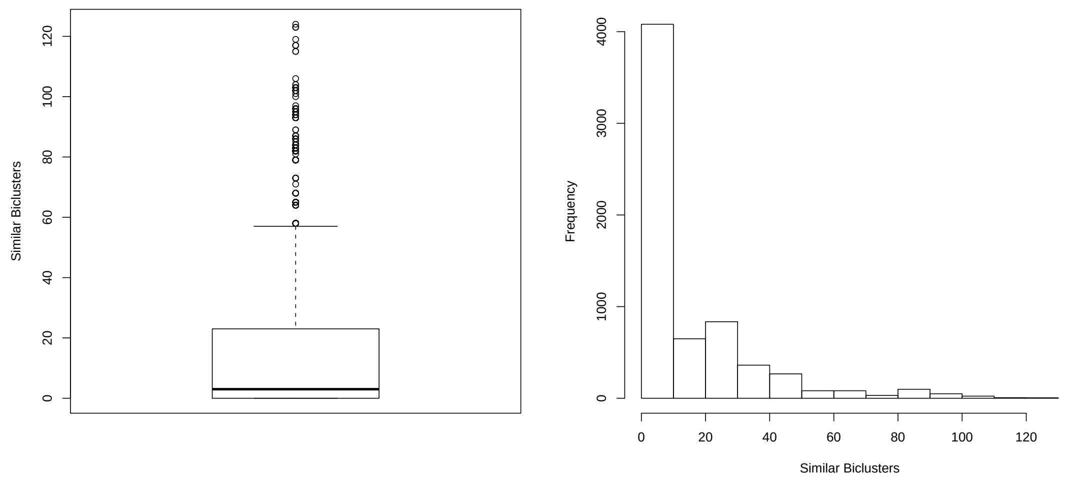

Figure 5.3.: Number of biclusters marked as similar for each observed bicluster. $min = 0$; $25\% - quantile = 0$; $median = 3$; $mean = 13.74$; $75\% - quantile = 23$; $max = 124$

5.3.2. Jaccard Index Approach

The threshold for the Jaccard Index was set to 0.9 as it allows nearly the same divergence between two biclusters as with the correlation approach (see section 4.2). Thus biclusters with a Jaccard Index greater than 0.9 were marked as similar. The quantity of similar biclusters according to this threshold is shown in figure 5.4, the extremely positively skewed distribution again implies that there are some biclusters which were found above average.

Again, only the upper $25\% - quantile$ of biclusters were kept, which this time leads to 63 remaining biclusters. The size of the biclusters varied in the gene dimension between 2 and 443 ($median = 141$; $mean = 147.8$) and in the sample dimension from 28 to 52 ($median = 41$; $mean = 40.4$) which is in fact quite similar to the results obtained from the correlation approach. The biclusters were found

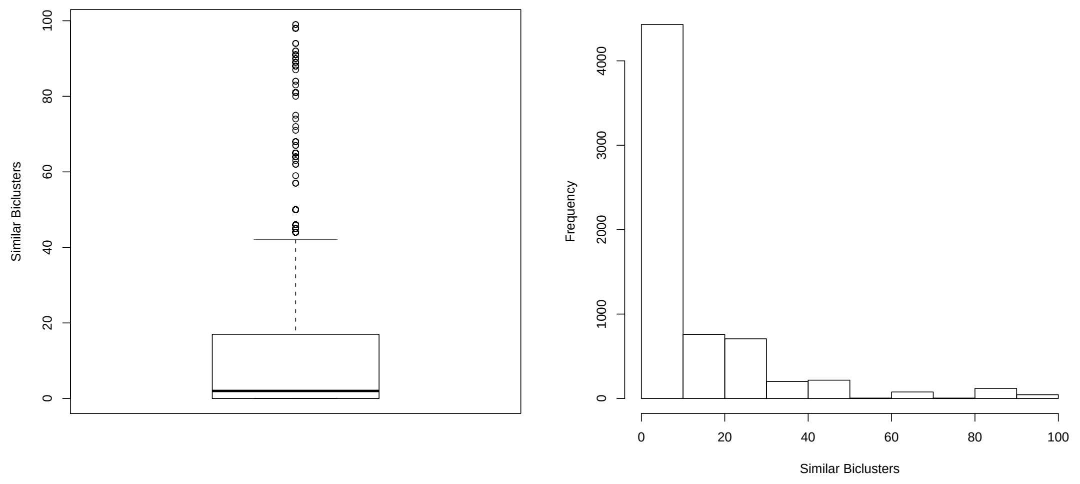

Figure 5.4.: Number of biclusters marked as similar for each observed bicluster. $min = 0$; $25\% - quantile = 0$; $median = 2$; $mean = 11.14$; $75\% - quantile = 17$; $max = 99$

between 19 and 89 times ($median = 25$; $mean = 29.60$). Furthermore the unique-gene-rate mentioned in the last section is also nearly the same (10.88%). Here too, the bicluster-score distribution can be found in figure 5.5. By comparing both scores obtained with the different methods it seems that they have a pretty similar, although shifted distribution. Which brings us to the question if there are any differences in the results of both approaches? This issue will be discussed in the next section.

5.3.3. Results in Comparison

Since both methods described above aim to obtain the best biclusters, it is of great interest whether they really lead to the same results or not. That is to the same biclusters. In order to get the similarity between the two results the obtained biclusters were again compared with the Jaccard Index and a threshold value of 0.9.

In total there were 43 biclusters which were found in each method. Which makes a similarity of 68.25% with reference to the results obtained by the Jaccard Index approach and a similarity of 74.14% with reference to the correlation approach results. When looking at the biclusters which were not observed with both methods it turned out that their average score of 0.021 (Jaccard Index approach) is

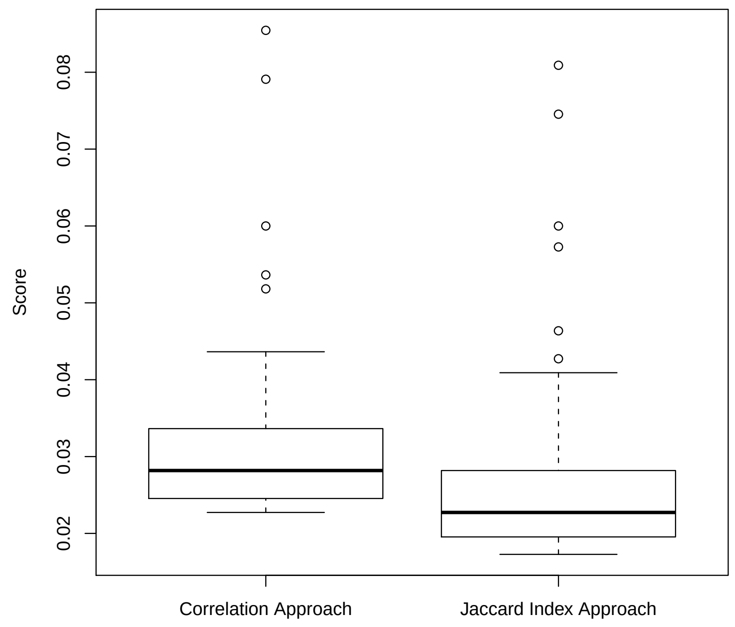

Figure 5.5.: Distribution of the bicluster scores. Correlation approach: $min = 0.023$; $25\% - quantile = 0.025$; $median = 0.028$; $mean = 0.033$; $75\% - quantile = 0.034$; $max = 0.085$. Jaccard Index approach: $min = 0.017$; $25\% - quantile = 0.020$; $median = 0.023$; $mean = 0.027$; $75\% - quantile = 0.028$; $max = 0.81$.

beneath the median (0.023) and the mean (0.027) total score. In contrast, the average score of biclusters which were obtained with each method is with 0.029 even above the $75\% - quantile$ (0.028) of the total score. In other words, the proposed score indeed seem to provide information about the quality of a obtained bicluster. Figure 5.6 shows a comparison of the two different score distributions which indeed indicates that biclusters found with both methods have a higher score than biclusters just found with one method.

It can be concluded that both methods, for the most part, seem to have marked the same biclusters as the best ones since their results overlap in a huge way. Second, the introduced score holds information about the goodness of an observed bicluster. However, based on these analyses it could not be concluded which of the introduced methods works better, thus is more powerful.

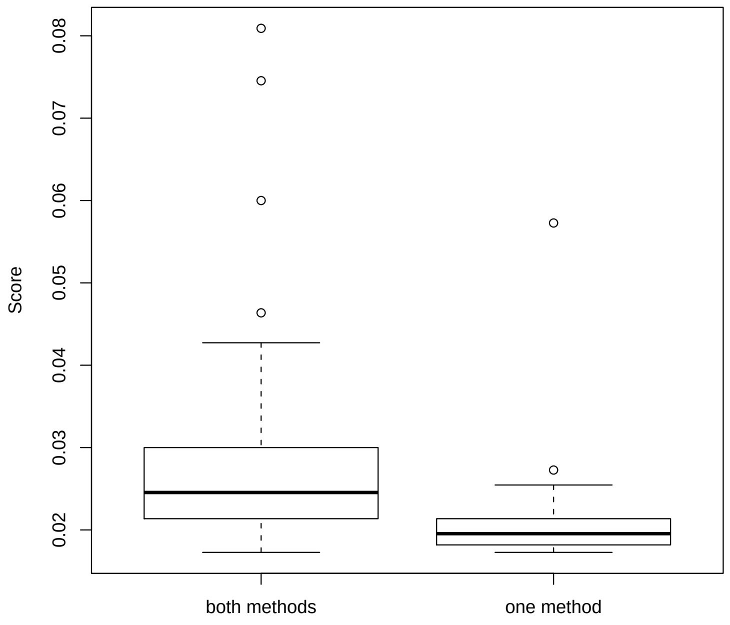

Figure 5.6.: Score of biclusters observed with only one method: $min = 0.017$; $25\% - quantile = 0.018$; $median = 0.019$; $mean = 0.022$; $75\% - quantile = 0.021$; $max = 0.057$. Observed with both methods: $min = 0.017$; $25\% - quantile = 0.021$; $median = 0.025$; $mean = 0.029$; $75\% - quantile = 0.030$; $max = 0.81$.

5.3.4. Conclusion on the right Release Levels

Even though the proposed ensemble methods appear to be able to find good biclusters without carrying about the `row.release` and `col.release` levels there might still be the question whether there is an optimal release level for all possible situations. Or rather, if all observed biclusters (after applying the ensemble method) were generated by the same release level or if they were generated by different levels. Maybe a certain level is only be able to find a particular type of bicluster, so that the observed biclusters vary in some specific attributes (e.g. the bicluster size). Besides, the proposed method requires an high computational demand since the plaid model has to be computed several times, which could be avoided in the case one knows optimal release levels. Given these observed biclusters it is therefore from great interest to draw conclusions on their `row.release` and `col.release` levels.

All the following analyses are only based on the results generated by the Jaccard Index approach. The correlation approach results are disregarded. Again, the R-Code is given in Appendix D as well as on the attached CD. An obvious start in analysing the release levels is to figure out which release settings have generated the remaining 63 biclusters. It turned out, that every release level has contributed some biclusters to the final result. However, small values tend to appear more often than large values. Indeed, this could be just a matter of the fact that small levels in general seem to obtain more biclusters than large levels. That means, just by looking at how often a certain release level appears in the results one is not able to get deeper information about appropriate values. Figure 5.7 shows the number of observed biclusters for each release after the ensemble method had been applied and figure 5.8 illustrates the total number of biclusters obtained by the different release levels. As already mentioned, both graphs illustrate, that less biclusters have been observed, the larger the release value is.

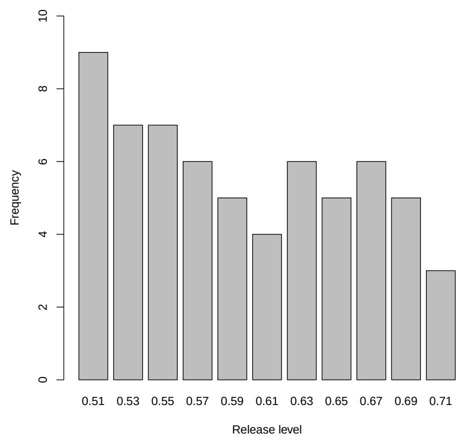

Figure 5.7.: Number of resulting (after applying the ensemble method) biclusters obtained by each `row.release` and `col.release` level.

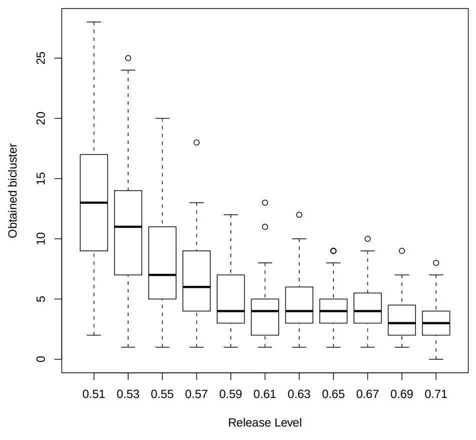

Figure 5.8.: Number of biclusters obtained by each `row.release` and `col.release` level in total.

In a next step one might want to know how the proposed score behaves amongst the different `row.-` and `col.release` level. In other words, whether there is a particular level with a significant high score or not. However, no conclusions could be drawn from considering the distribution of the score. It rather appears, that the different score values are randomly spread over the release levels.

Since it seems that every release level has contributed its part to the result the question arises whether there are any differences in biclusters obtained with different release values or not. For this reason, one can look at the size of the biclusters, that is the row, the column and the total ($rows * columns$) size. Looking at figure 5.9 suggests, that there are no differences in the column sizes, but in the row sizes as well as in the total sizes. Whereas the variation in size decreases with increasing release values. Although the median is highest in the middle and lower at the ends the mean size also decreases as soon as the release level increases. Which of course is due to the extremely right skewed distribution at smaller release levels. This finding implies, that there are indeed differences in the kind of biclusters a certain release value is able to find.

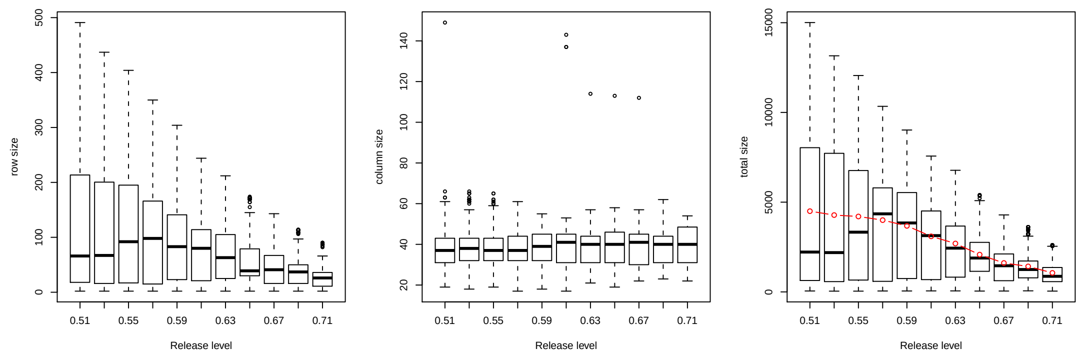

Figure 5.9.: The size of the biclusters against the different release levels separate for rows, columns and the total ($rows * columns$) size. The red points in the right graph symbolises the mean total size.

In order to verify the assumption that each release level finds a different kind of biclusters one can look at the consistence of the observed biclusters within and between the release levels. More precisely, at the number of similarly marked biclusters relative to their `row.release` and `col.release` values. In fact, the diagonal structure of table 5.4 very clearly shows that a bicluster obtained with a particular release level is besides only obtained by values close to this certain value (i.e. the one above or below). In other words, a bicluster observed with a certain release is very unlikely to be observed with a totally different value. One could wonder if this still holds true when looking at the similarity of each dimension separately (therefore the Jaccard Index had to be computed for each dimension separately). It turned out that the similarity amongst bicluster rows obtained with different release levels behaves the same. That is, bicluster genes observed with a certain value are unlikely to be observed with an other value. By contrast, however, the similarity of the bicluster columns does not depend on the `row.release` and `col.release` level. Although the probability of obtaining similar bicluster samples decreases the further the release levels are apart (the corresponding tables can be found in appendix C). The completely different genes (rows) and samples (columns) sizes of the data matrix might be a explanation for this effect, since there are way more possibilities to form a cluster in a vector of the length 12042 (row size) than in a vector of the length 202 (column size).

	`0.51`	`0.53`	`0.55`	`0.57`	`0.59`	`0.61`	`0.63`	`0.65`	`0.67`	`0.69`	`0.71`
`0.51`	11124	73	0	0	0	2	0	0	0	0	0
`0.53`	73	10634	666	0	0	0	0	0	0	0	0
`0.55`	0	666	6672	16	0	0	0	0	0	0	0
`0.57`	0	0	16	7954	78	0	0	0	0	0	0
`0.59`	0	0	0	78	5926	3	0	0	0	0	0
`0.61`	2	0	0	0	3	4348	47	0	0	0	0
`0.63`	0	0	0	0	0	47	4932	24	1	0	0
`0.65`	0	0	0	0	0	0	24	5728	497	0	0
`0.67`	0	0	0	0	0	0	1	497	5530	178	0
`0.69`	0	0	0	0	0	0	0	0	178	3572	158
`0.71`	0	0	0	0	0	0	0	0	0	158	3246

Table 5.4.: The table shows the number of similar marked biclusters between each release level combination. For example 73 biclusters found with a `row.-` and `col.release` level of 0.51 (0.53 respectively) were also obtained with a release value of 0.53 (0.51 respectively).

There is the possibility, that two biclusters have not been marked as similar according to one of the proposed similarity measures although either of them contains all elements of the other bicluster. Consider the case that a bicluster is a small subset (i.e. submatrix) of another bicluster but their Jaccard Index has not exceeded the threshold value. Under this circumstances, these two biclusters are not marked as similar although they are obviously similar even though with different dimensions. That means, although there are no biclusters marked as similar between two `release` levels which are far apart, nevertheless there could be potentially similar biclusters between these two values. To see whether bicluster results obtained with large release levels (which in fact tend to find smaller bicluster) are subsets of biclusters observed with smaller values one has to apply an slightly modified Jaccard Index. Given two bicluster A, B and $|A| > |B|$ the index is defined as follows:

$$\tilde{J}(A, B) = \frac{|A \cap B|}{|B|}. \tag{5.1}$$

Hence, a value of 1 means that bicluster B is completely included in bicluster A. Results obtained with release levels of 0.59, 0.61 and 0.69 were then again compared with this adjusted Jaccard Index. But in this case only biclusters with an adjusted Jaccard value of 1 were marked as similar. As one can see from table 5.5 there is indeed a huge amount of biclusters observed with a level of 0.69 which are submatrices of biclusters obtained with a level of 0.59 or 0.61. Bear in mind, by applying the normal Jaccard Index to these biclusters there were no biclusters marked as similar between 0.69 and 0.59 or 0.61 and just 3 between 0.59 and 0.61 (see table 5.4). This suggests, nevertheless there are no accordances between two release levels which are far apart with the original Jaccard Index, there are still similar biclusters within these levels. That means, different release levels do not obtain totally different biclusters, in fact most biclusters obtained by large release values seem to form submatrices of biclusters observed with smaller values since they tend to be smaller in size.

	`0.59`	`0.61`	`0.69`
`0.59`	3742	2502	1239
`0.61`	2502	3686	953
`0.69`	1239	953	3602

Table 5.5.: The table shows the number of similar marked biclusters between the three release levels according to a adjusted Jaccard Index of 1.

In summary one can say that each `row.release` and `col.release` level has contributed a part to the final bicluster results. In addition, it appeared that the size of the observed biclusters is dependent on the release level. More precisely, the larger the release level, the smaller the observed bicluster. To specify the less genes are included in a bicluster, since the number of included samples does not seem to be dependent on the release level. However, it also turned out that there is a huge ratio of biclusters which are submatrices of biclusters obtained with a different release value. That means, the different levels do not obtain completely divers biclusters, for the most part the results just vary in size.

6. Conclusions and Discussion

This thesis deals with the application of bicluster algorithms on real gene expression data. Several different methods were studied and validated based on the bicluster algorithm, the bicluster structure (exclusive rows/columns, non-overlapping non exclusive, arbitrarily positioned) and the bicluster type (constant values/rows/columns, coherent values and coherent evolution) each method is able to find. Given these facts, the plaid algorithm was chosen to fit the requirements for the application to genomics data best.

The plaid algorithm, however, has some parameters, which values are free to choose. Through applying the method to simulated, as well as real data has been shown, that the results are highly depending on the input parameters, more precisely on the chosen `row.release` and `col.release` levels. Moreover, optimal release values are not known and even with identical parameter settings the algorithm is likely to observe different biclusters in different runs. To overcome these restrictions an ensemble method has been proposed in this work.

The introduced ensemble methods are based on the idea that when applying the plaid algorithm several times with varying release values the real bicluster are more likely to be found more often than random bicluster. To compare two biclusters regarding their similarity two methods have been proposed. One of which makes use of the Jaccard Index and one of the pairwise correlation between these biclusters. In order to get the number of similar biclusters, thus the best bicluster one has to set threshold values for the Jaccard Index, respectively correlation. To figure out which threshold values lead to the best results could be interest of further studies. For example, it would be good if the correlation threshold would depend on the size of the data matrix as well as the size of the bicluster since these two values have an influence on the correlation. However, this goes beyond this thesis, here the thresholds were set to fixed values. In addition to the ensemble methods a score is proposed which rates the resulting bicluster based on their frequency.

By looking at the results (i.e. the obtained bicluster) it turned out that both methods have found to a large extent the same bicluster. However, based on these results no conclusions on the optimal `row.release` and `col.release` values could be drawn. Instead, it appeared that different levels do obtain similar bicluster, although the size of the biclusters varies amongst the different values. Large release levels tend to obtain smaller biclusters than smaller values. It might be meaningful to present the obtained biclusters to biologists in order to see whether the method has lead to already known and/or biological relevant results.

As mentioned above, further research is required on the choice of optimal threshold values for the similarity measures (i.e. Jaccard Index, respectively correlation). Furthermore, it might be of interest to see how the results change when `row.release` and `col.release` are not set equal in each run but rather vary. That means when a combination of small and large release values is applied to the data. All in all in the area of biclustering is still a tremendous need for further and deeper studies as it is a relatively new field.

A. Bibliography

S. Barkow, S. Bleuler, A. Prelic, P. Zimmermann, and E. Zitzler. BicAT: A Biclustering Analysis Toolbox. *Bioinformatics*, 22:1282–1283, 2006.

S. Bergmann, J. Ihmels, and N. Barkai. Iterative signature algorithm for the analysis of large-scale gene expression data. *Physical Review*, 67(3), March 2003.

Y. Cheng and G. Church. Biclustering of expression data. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, number 1, pages 93–103. American Association for Artificial Intelligence, 2000.

G. Getz, E. Levine, and E. Domany. Coupled two-way clustering analysis of gene microarray data. *Proc Natl Acad Sci U S A*, 97(22):12079–12084, October 2000.

J. A. Hartigan. Direct clustering of a data matrix. *American Statistical Association*, 67(337):123–129, 1972.

J. Ihmels, G. Friedlander, S. Bergmann, O. Saring, Y. Ziv, and N. Barkai. Revealing modular organization in the yeast transcriptional network. 2003.

P. Jaccard. Distribution de la flore alpine dans le bassin des dranses et dans quelques règions voisines. *Bulletin de la Sociate Vaudoise des Sciences Naturelles*, 37:241–272, 1901.

Sebastian Kaiser and Friedrich Leisch. A Toolbox for Bicluster Analysis in R. In *Compstat 2008 — Proceedings in ComputationalStatistics*. Physica Verlag, 2008.

Sebastian Kaiser, Rodrigo Santamaria, Martin Sill, Roberto Theron, Luis Quintales, and Friedrich Leisch. *biclust: BiCluster Algorithms*, 2009. URL `http://R-Forge.R-project.org/projects/biclust/`. R package version 0.9.1/r158.

Y. Kluger, R. Basri, J. T. Chang, and M. Gerstein. Spectral biclustering of microarray data: Coclustering genes and conditions. *Genome Res*, 13(4):703–716, April 2003.

L. Lazzeroni and A. Owen. Plaid Models for Gene Expression Data. *Statistica Sinica*, 12:61–86, 2000.

S. C. Madeira and A. L. Oliveira. Biclustering Algorithms for Biological Data Analysis: A Survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):24–45, 2004.

R. McLendon, A. Friedman, D. Bigner, E.G. Van Meir, D.J. Brat, G.M. Mastrogianakis, J.J. Olson, T. Mikkelsen, N. Lehmann, K. Aldape, W.K.A. Yung, O. Bogler, J.N. Weinstein, S. VandenBerg, M. Berger, and Prados. Comprehensive genomic characterization defines human glioblastoma genes and core pathways. *Nature*, 455:1061–1068, 2008.

T. M. Murali and S. Kasif. Extracting Conserved Gene Expression Motifs from Gene Expression Data. *Proceedings of the 8th Pacific Symposium on Biocomputing*, 8:77–88, 2003.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

A. Tanay, R. Sharan, and R. Shamir. Biclustering Algorithms: A Survey. In *Handbook of Computational Molecular Biology / CRC Computer and Information Science Series*, 2005.

Stefan Theußl and Achim Zeileis. Collaborative Software Development Using R-Forge. *The R Journal*, 1(1):9–14, May 2009. URL `http://journal.r-project.org/2009-1/RJournal_2009-1_Theussl+Zeileis.pdf`.

B. Algorithms Summary

	Algorithm	Type of Bicluster	Structure	excl. noise
1	Block Clustering	Constant	Exclusive Rows or Columns	No
2	δ-Method	Coherent Values	Overlapping (but not likely)	Yes
3	CTWC	Constant Columns	Arbitrarily positioned	Yes
4	ISA	Coherent Values	Arbitrarily positioned	Yes
5	Spectral	Coherent Values	Checkerboard like	No
6	xMotifs	Coherent Evolution	Arbitrarily positioned	Yes
7	Plaid Models	Coherent Values	Arbitrarily positioned	Yes
8	GRASP	Coherent Values	Overlapping (but not likely)	Yes
9	pClusters	Coherent Values	Exclusive Rows or Columns	No
10	FLOC	Coherent Values	Arbitrarily positioned	Yes
11	ITWC	Coherent Values	Exclusive Rows or Columns	No
12	Gibbs	Constant Columns	Exclusive Rows or Columns	No
13	PRMs	Coherent Values	Arbitrarily positioned	Yes
14	Enigma	Coherent Evolution	Arbitrarily positioned	Yes
15	VOTE	Coherent Values	Arbitrarily positioned	Yes
16	DCC	Constant	Checkerboard like	No
17	OPSMs	Coherent Evolution	Arbitrarily positioned	Yes
18	δ-Patterns	Constant Rows	Arbitrarily positioned	Yes
19	BBC	Coherent Values	Exclusive Rows or Columns	Yes
20	QUIBIC	Coherent Evolution	Arbitrarily positioned	Yes
21	SAMBA	Coherent Evolution	Arbitrarily positioned	Yes
22	OP-Clusters	Coherent Evolution	Arbitrarily positioned	No
23	LAS	Constant	Arbitrarily positioned	Yes
24	MSB	Coherent Values	Arbitrarily positioned	Yes

C. Similarity of Biclusters

	0.51	0.53	0.55	0.57	0.59	0.61	0.63	0.65	0.67	0.69	0.71
0.51	13546	1280	0	0	0	3	1	1	1	0	0
0.53	1280	13274	1202	0	0	0	0	0	0	0	0
0.55	0	1202	9206	426	0	0	0	0	0	0	0
0.57	0	0	426	8792	191	0	0	0	0	0	0
0.59	0	0	0	191	6742	53	0	0	0	0	0
0.61	3	0	0	0	53	5072	142	7	3	0	0
0.63	1	0	0	0	0	142	6282	25	2	0	0
0.65	1	0	0	0	0	7	25	6134	922	470	0
0.67	1	0	0	0	0	3	2	922	5786	798	10
0.69	0	0	0	0	0	0	0	470	798	4622	239
0.71	0	0	0	0	0	0	0	0	10	239	4200

Table C.1.: The table shows the number of similar marked bicluster rows between each release level combination.

	0.51	0.53	0.55	0.57	0.59	0.61	0.63	0.65	0.67	0.69	0.71
0.51	22492	20524	17025	11444	9800	7999	8194	8084	4745	3603	939
0.53	20524	20774	17933	11971	10041	8109	8407	8154	5174	4073	1280
0.55	17025	17933	17730	14021	11523	7742	7742	7544	4414	3018	158
0.57	11444	11971	14021	16276	13366	10836	9919	9912	5862	4018	1329
0.59	9800	10041	11523	13366	12296	10961	9058	9218	6943	4065	1698
0.61	7999	8109	7742	10836	10961	12572	11278	10768	10115	6436	1956
0.63	8194	8407	7742	9919	9058	11278	11740	11395	9761	6085	1944
0.65	8084	8154	7544	9912	9218	10768	11395	13508	12123	5904	2262
0.67	4745	5174	4414	5862	6943	10115	9761	12123	14618	8159	3939
0.69	3603	4073	3018	4018	4065	6436	6085	5904	8159	7660	4954
0.71	939	1280	158	1329	1698	1956	1944	2262	3939	4954	6750

Table C.2.: The table shows the number of similar marked bicluster columns between each release level combination.

D. R-Code and Bash-Script

D.1. jobNewR

The following bash script has been running on a cluster computer setup for each release level 100 times. Whereas the script below shows an example for a release value of 0.51. The script had to be adjusted for each release level in order that the resulting files do not overwrite each other. See attached CD for all files.

```
#!/bin/bash
#
#$ -clear
#$ -q bigMem.q,default.q
#
# Set shell for job
#$ -S /bin/bash
#
# Execute job from current working directory
#$ -cwd
#
# merge std error and std out into one file
#$ -j y
#

R="/home/socci/bin/R"

hostname
pwd

logfile="runlogTCGA051__"$SGE_TASK_ID
echo $logfile

time $R CMD BATCH --vanilla $1 $logfile

echo $SGE_TASK_ID, "DONE"
```

D.2. runPlaid.r

The R file below has been applied in combination with the bash script mentioned above in order to run the plaid model on the data. Again, the file is just an example for the release level 0.51 and had to be adjusted for the rest.

```
install.packages("biclust")
library(biclust)

# read data and set gene and sample vectors
daNO <- read.table("data/TCGAdata_continues.txt",
                   header=FALSE, row.names=1, nrows=1)
sample.types <- daNO[1,]
daNO <- read.table("data/TCGAdata_continues.txt",
                   header=TRUE, row.names=1)
daNO <- as.matrix(daNO)
gene.names <- row.names(daNO)

# set row and col release levels
ROWR <- COLR <- 0.51

# get running number of the computer cluster system
taskID <- as.numeric(Sys.getenv()[["SGE_TASK_ID"]])

# apply plaid model on the data
RE <- biclust(x=daNO, method=BCPlaid(), cluster="b",
              fit.model=y~m+a+b, background=TRUE,
              row.release=ROWR, col.release=COLR,
              shuffle =3, back.fit=0, max.layers=100,
              iter.startup=15, iter.layer=30, verbose=TRUE)

#save the results
save(RE, file=paste("out",ROWR,"_", taskID, ".Rdata",
                    sep= ""))
```

D.3. combine_biclust.r

The functions below combine the results in order to make further computations easier and to reduce the required hard disc space. The combination had to be split in two functions due to a too large RAM usage.

```
# function to combine objects of the class biclust
##################################################

combine.results <- function(wd, thr=11, sim=100) {

  # set wd to directory which includes the results
  setwd(wd)
  files <- dir()
  ll <- thr
  l <- sim

  RES <- vector("list", length = ll)
  helper <- vector("list", length = l)

  # loop through the biclust results
  for (mm in (1:ll)) {
    rowr <- 0.51 + 0.02 * (mm - 1)
    RES[[mm]] <- helper
    for (m in (1:l)) {
      load(paste(files[mm], "/out", rowr, "_", m,
                 ".Rdata", sep= ""))
      # to reduce required disc space
      RE@Parameters$Data <- NULL
      RES[[mm]][[m]] <- RE
    }
  }

  comb.RES <- RES
  save(comb.RES, file="combined_results.RData")
  comb.RES
}


# function to combine all obtained biclusters
# after applying the function above in order to
# get an overall result matrix as well as
# vectors including the size of the biclusters
##################################################
```

```
combine.clust <- function(comb.RES) {

  rl <- length(comb.RES)
  rowsnew <- c()
  colsnew <- c()
  rsize <- c()
  csize <- c()

  # loop through results
  for (i in 1:rl) {
    rll <- length(comb.RES[[i]])
    for (j in 1:rll) {
      # check if biclusters have been found
      if(comb.RES[[i]][[j]]@Number>0) {
        # combine biclusters
        lee <- nrow(comb.RES[[i]][[j]]@NumberxCol)
        rowsnew <- cbind(rowsnew, comb.RES[[i]]
                         [[j]]@RowxNumber)
        colsnew <- cbind(colsnew, t(comb.RES[[i]]
                         [[j]]@NumberxCol))
        for(t in 1:lee) {
          # get dimension of biclusters
            rsize <- c(rsize, sum(comb.RES[[i]][[j]]
                              @RowxNumber[,t]))
            csize <- c(csize, sum(comb.RES[[i]][[j]]
                              @NumberxCol[t,]))
        }
      }
    }
  }

  comb.clust <- list(rowsnew, colsnew, rsize, csize)
  save(comb.clust, file="combined_clust.RData")
  comb.clust
}
```

D.4. jaccard.r

The following functions are partly based on the Jaccard Index function `jaccardind` of the package `biclust` by Kaiser and Leisch [2008].

```
# function to compute the original jaccard index
##################################################

jaccard <- function(resr, resc){

  jaccardmat <- matrix(nrow=ncol(resr), ncol=ncol(resr))

  # loop through all pairwise bicluster combinations
  for(i in 1:ncol(resr)) {
    # get bicluster 1
    alle1<-resr[,i] %*% t(resc[,i])
    for(j in i:ncol(resr)) {
      # get bicluster 2
      alle2<-resr[,j] %*% t(resc[,j])
      # get union
      alle<-alle1 + alle2
      loalle<-alle>0
      loalle1<-alle1>0
      loalle2<-alle2>0
      jaccardmat[i,j]<- (sum(loalle1)+sum(loalle2)-
                         sum(loalle))/sum(loalle)
    }
    print(paste("Bicluster", i, "done!"))
  }
  jaccardmat
}


# function to compute the jaccard index separate
# for the rows and columns
##################################################

jaccard.sep <- function(resr, resc){

  matrow <- matrix(nrow=ncol(resr), ncol=ncol(resr))
  matcol <- matrix(nrow=ncol(resr), ncol=ncol(resr))

  # loop through all pairwise bicluster combinations
  for(i in 1:ncol(resr)) {
    alle1 <- resr[,i]>0
    alle11 <- resc[,i]>0
    for(j in i:ncol(resr)) {
      alle2 <- resr[,j]>0
      alle22 <- resc[,j]>0
      alle <- alle1 + alle2
      alle12 <- alle11 + alle22
      loalle <- alle>0
      loalle12 <- alle12>0
      # calculate jaccard index for rows and cols
      matrow[i,j] <- ((sum(alle1)+sum(alle2)-sum(loalle))
                       /sum(loalle))
      matcol[i,j] <- ((sum(alle11)+sum(alle22)-
                       sum(loalle12))/sum(loalle12))
    }
    print(paste("Bicluster", i, "done!"))
  }
  list(matrow, matcol)
}


# function to compute the adjusted jaccard index
#################################################

jaccard.adj <- function(resr, resc) {

  jaccardmat <- matrix(nrow=ncol(resr), ncol=ncol(resr))

  # loop through all pairwise bicluster combinations
  for(i in 1:ncol(resr)) {
    # get bicluster 1
    clust1 <- resr[,i] %*% t(resc[,i])
    for(j in i:ncol(resr)) {
      # get bicluster 2
      clust2 <- resr[,j] %*% t(resc[,j])
      # check which bicluster is smaller
      if(sum(clust1)>sum(clust2)) {
        jaccardmat[i,j] <- (sum(clust2[clust1==clust2]>0)
                            /sum(clust2))
        }
        else {
          jaccardmat[i,j] <- (sum(clust1[clust1==clust2]>0)
                              /sum(clust1))
          }
      }
      print(paste("Bicluster", i, "done!"))
  }
  jaccardmat
}
```

D.5. correlation.r

```
# function to calculate the correlation between
# the biclusters separate for each dimension
##################################################

get.cor <- function(RES) {

  rowsnew <- RES[[1]]
  colsnew <- RES[[2]]
  rsize <- RES[[3]]
  csize <- RES[[4]]

  cmr <- cor(rowsnew)
  cmc <- cor(colsnew)
  diag(cmr) <- 0
  diag(cmc) <- 0

  ClustCor <- list(rowsnew=rowsnew, colsnew=colsnew,
                   rsize=rsize, csize=csize, cmr, cmc)
  save(ClustCor, file="ClustCor.RData")
  ClustCor
}
```

D.6. ensemble_method.r

The functions below provide a way to apply the proposed ensemble methods to pre-computed correlation-, respectively Jaccard Index matrices.

```
# function to get and plot the number of similar
# biclusters with the Jaccard Index method
##################################################

get.rowsum.jac <- function(rsize, csize, jac, thr=0.9,
                           pdf=FALSE) {

  # check if bicluster should be marked as similar
  uni <- jac>thr
  # count number of similarly marked bicluster
  rowsum <- rowSums(uni)

  # plot number of similarly marked bicluster
  if(pdf==TRUE) {
    pdf("SimBox_jac.pdf")
    boxplot(rowsum, ylab="Similar Biclusters")
    dev.off()
    pdf("SimHist_jac.pdf")
    hist(rowsum, xlab="Similar Biclusters", main="")
    dev.off()
    }

  list(rowsum, uni)
}


# function to get and plot the number of similar
# bicluster with the correlation method
##################################################

get.rowsum.cor <- function(rsize, csize, cmr, cmc,
                           thr=c(0.95,0.9), pdf=FALSE) {

  # check if bicluster should be marked as similar
  uni.r <- cmr>thr[1]
  uni.c <- cmc>thr[2]
  unihelp <- uni.r+uni.c
  uni <- unihelp==2
  # count number of similarly marked bicluster
  rowsum <- rowSums(uni)

  # plot number of similarly marked bicluster
  if(pdf==TRUE) {
    pdf("SimBox_cor.pdf")
    boxplot(rowsum, ylab="Similar Biclusters")
    dev.off()
    pdf("SimHist_cor.pdf")
    hist(rowsum, xlab="Similar Biclusters", main="")
    dev.off()
  }

  list(rowsum, uni)
}


# function to get the best bicluster according
# to the proposed ensemble method and their score
##################################################

excl.bicluster <- function(rowsum, uni, rowsnew,
                           quant=0.75) {

  # excl. a certain quantile of bicluster
  excl <- quantile(rowsum, quant)
  for (y in 0:excl) {
    rowsum[rowsum==y] <- FALSE
    }

  rowsum[rowsum!=FALSE] <- TRUE
  clustind <- rowsum

  # excl. similar bicluster
  for (m in 1:length(rowsum)) {
    if (clustind[m]==1) {
      helpind <- which(uni[m,]==TRUE)
      clustind[helpind] <- FALSE
    }
  }

  # get index of results
  clustindtotal <- as.logical(clustind)

  # compute bicluster score
  score <- (rowSums(uni[clustindtotal, ])+1)/1100
  print(paste("No. of remaining cluster:",
              sum(clustindtotal)))
  return(list(clustindtotal, score))
}


# function to print and get the results in a
# proper way
##################################################

print.bicluster <- function(clustindtotal, score, rowsnew,
                            colsnew, gene.names,
                            sample.types, print=TRUE) {

  helpind <- which(clustindtotal==TRUE)
  ll <- length(helpind)

  genes <- vector(mode="list", length=ll)
  gsize <- c()
  samples <- vector(mode="list", length=ll)
  ssize <- c()
  gene.vector <- c()

  # loop through results
  for (i in 1:ll) {
    # get genes/samples/size of resulting bicluster
    gene.ind <- rowsnew[,helpind[i]]
    sample.ind <- colsnew[,helpind[i]]
    gene.help <- gene.names[gene.ind]
    gene.vector <- c(gene.vector, gene.help)
    sample.help <- sample.types[sample.ind]
    gsizehelp <- colSums(rowsnew[,clustindtotal])[i]
    ssizehelp <- colSums(colsnew[,clustindtotal])[i]
    gsize <- c(gsize, gsizehelp)
    ssize <- c(ssize, ssizehelp)
    genes[[i]] <- gene.help
    samples[[i]] <- sample.help

          # print resulting bicluster
    if(print==TRUE) {
      print(" ")
      print(paste("CLUSTER",i))
      print(" ")
      print(paste("Score:", score[i]))
      print(" ")
      print(paste(gsizehelp, "genes and", ssizehelp,
                  "samples"))
      print(" ")
      print("Genes:")
      print(genes[[i]])
      print(" ")
      print("Samples:")
      print(samples[[i]])
      print(" ")
      print("-----------------------------------------
            ----------------------")
          }
  }

  # print highest/lowest score and summary
  if(print==TRUE) {
    print(" ")
    print("Cluster with the highest score:")
    print(which(score==max(score)))
    print("Cluster with the lowest score:")
    print(which(score==min(score)))
    print("Score summary:")
    print(summary(score))
    print(" ")
    print("-------------------------------------------
          --------------------")
  }

  list(gsize, ssize, genes, samples)
}


# final function which combines the functions
# above, for the correlation method
##################################################

get.bicluster.cor <- function(ClustCor, gene.names,
                              sample.types, quant=0.75,
                              thr=c(0.95,0.9),
                              print=TRUE, boxplot=FALSE) {

  rowsnew <- ClustCor[[1]]
  colsnew <- ClustCor[[2]]
  rsize <- ClustCor[[3]]
  csize <- ClustCor[[4]]
  cmr <- ClustCor[[5]]
  cmc <- ClustCor[[6]]

  # get number of similar bicluster
  help <- get.rowsum.cor(rsize, csize, cmr, cmc,
                         thr, pdf=boxplot)
  rowsumhelp <- help[[1]]
  uni <- help[[2]]
  # excl. bad and similar bicluster
  Clusterhelp <- excl.bicluster(rowsumhelp, uni,
                                rowsnew=rowsnew, quant=quant)
  clustindtotal <- Clusterhelp[[1]]
  score <- Clusterhelp[[2]]

  # print and get results
  Print <- print.bicluster(clustindtotal, score, rowsnew,
                           colsnew, gene.names, sample.types,
                           print=print)

  list(genes=Print[[3]], samples=Print[[4]], score=score,
       clustindtotal=clustindtotal, rowsumhelp=rowsumhelp,
       gsize=Print[[1]], ssize=Print[[2]])
}


# final function which combines the functions
# above, for the Jaccard Index method
##################################################

get.bicluster.jac <- function(ClustJac, gene.names,
                              sample.types, quant=0.75,
                              thr=0.9, print=TRUE,
                              boxplot=FALSE) {

  rowsnew <- ClustJac[[1]]
  colsnew <- ClustJac[[2]]
  rsize <- ClustJac[[3]]
  csize <- ClustJac[[4]]
  jac <- ClustJac[[5]]

  # get number of similar bicluster
  help <- get.rowsum.jac(rsize, csize, jac, thr,
                         pdf=boxplot)
  rowsumhelp <- help[[1]]
  uni <- help[[2]]
  # excl. bad and similar bicluster
  Clusterhelp <- excl.bicluster(rowsumhelp, uni,
                                rowsnew=rowsnew, quant=quant)
  clustindtotal <- Clusterhelp[[1]]
  score <- Clusterhelp[[2]]

  # print and get results
  Print <- print.bicluster(clustindtotal, score, rowsnew,
                           colsnew, gene.names, sample.types,
                           print=print)

  list(genes=Print[[3]], samples=Print[[4]], score=score,
       clustindtotal=clustindtotal, rowsumhelp=rowsumhelp,
       gsize=Print[[1]], ssize=Print[[2]])
}
```

D.7. calculations.r

In this R-code the TCGA data set is applied to the ensemble method. In addition, the resulting biclusters are analysed and compared between the proposed methods.

```
# computing the best bicluster
##################################################

# load necessary functions
source("combine_biclust.r")
source("jaccard.r")
source("correlation.r")
source("ensemble_method.r")

# set setwd to the directory which includes the
# biclust results '051' to '071' and combine results
comb.RES <- combine.results("setwd", thr=11, sim=100)
RES <- combine.clust(comb.RES)

# compute the correlation and
# Jaccard Index between the bicluster
ClustCor <- get.cor(RES)
ClustJac_help <- jaccard(RES$rowsnew, RES$colsnew)

# make diagonal matrix
lower <- lower.tri(ClustJac_help, diag=FALSE)
ClustJac_help[lower] <- t(ClustJac_help)[lower]
diag(ClustJac_help) <- 0
ClustJac <- list(RES$rowsnew, RES$colsnew, RES$rsize,
                 RES$csize, ClustJac_help)

save(ClustCor, file="RData/ClustCor.RData")
save(ClustJac, file="RData/ClustJac.RData")

# read data and set gene and sample vectors
daNO <- read.table("data/TCGAdata_continues.txt",
                   header=FALSE, row.names=1, nrows=1)
sample.types <- daNO[1,]
daNO <- read.table("data/TCGAdata_continues.txt",
                   header=TRUE, row.names=1)
daNO <- as.matrix(daNO)
gene.names <- row.names(daNO)
```

```
# get best bicluster according to the proposed methods
ResCor <- get.bicluster.cor(ClustCor, gene.names,
                            sample.types, quant=0.75,
                            thr=c(0.95,0.9), print=TRUE,
                            boxplot=TRUE)
ResJac <- get.bicluster.jac(ClustJac, gene.names,
                            sample.types, quant=0.75,
                            thr=0.9, print=TRUE,
                            boxplot=TRUE)
save(ResCor, ResJac, file="RData/Results.RData")


# comparing both results
##################################################

# bicluster dimensions
summary(ResCor$gsize)
summary(ResCor$ssize)
summary(ResJac$gsize)
summary(ResJac$ssize)

# number of biclusters marked as similar
summary(ResCor$rowsumhelp)
summary(ResJac$rowsumhelp)

# score of the bicluster
Score <- c(ResCor$score, ResJac$score)
Approach <- c(rep("Correlation Approach",
                  length(ResCor$score)),
              rep("Jaccard Index Approach",
                  length(ResJac$score)))
score <- data.frame(Score, Approach)

pdf("images/score.pdf")
boxplot(score$Score~score$Approach, ylab="Score")
dev.off()

summary(ResCor$score)
summary(ResJac$score)

# unique gene rate
helpCor <- do.call("c", ResCor$genes)
ngenesCor <- length(helpCor)
ugenesCor <- length(unique(helpCor))
rateCor <- ugenesCor/ngenesCor
```

```
rateCor

helpJac <- do.call("c", ResJac$genes)
ngenesJac <- length(helpJac)
ugenesJac <- length(unique(helpJac))
rateJac <- ugenesJac/ngenesJac
rateJac

# most represented gene
nCor <- rep(NA,ugenesCor)
for (ii in 1:ugenesCor) {
        help <- which(helpCor==unique(helpCor)[ii])
        nCor[ii] <- length(help)
        }
mnCor <- max(nCor)
help2 <- which(nCor==mnCor)
gnameCor <- unique(helpCor)[help2]
mnCor
gnameCor

nJac <- rep(NA,ugenesJac)
for (ii in 1:ugenesJac) {
        help <- which(helpJac==unique(helpJac)[ii])
        nJac[ii] <- length(help)
        }
mnJac <- max(nJac)
help2 <- which(nJac==mnJac)
gnameJac <- unique(helpJac)[help2]
mnJac
gnameJac

# comparing the results with the Jaccard Index
rowsnew <- comb.clust[[1]]
colsnew <- comb.clust[[2]]
rsize <- comb.clust[[3]]
csize <- comb.clust[[4]]
rowsnewtotal <- cbind(rowsnew[,ResJac$clustindtotal],
                      rowsnew[,ResCor$clustindtotal])
colsnewtotal <- cbind(colsnew[,ResJac$clustindtotal],
                      colsnew[,ResCor$clustindtotal])

JacCompare <- jaccard(rowsnewtotal, colsnewtotal)
lower <- lower.tri(JacCompare, diag=FALSE)
JacCompare[lower] <- t(JacCompare)[lower]
diag(JacCompare) <- 0
```

```
save(JacCompare, file="RData/JacCompare.RData")

unitotal <- JacCompare>0.9

unitotal12 <- unitotal[1:63,64:121]
unitotal21 <- unitotal[64:121,1:63]

rowsumhelp12 <- rowSums(unitotal12)
rowsumhelp21 <- rowSums(unitotal21)
rowsumhelp12[rowsumhelp12!=0] <- 1
rowsumhelp21[rowsumhelp21!=0] <- 1
sum(rowsumhelp12)
sum(rowsumhelp21)

# compare score of biclusters found with both
# and just found with one method
summary(ResJac$score[which(rowsumhelp12==0)])
summary(ResJac$score[which(rowsumhelp12==1)])

ScoreC <- c(ResJac$score[which(rowsumhelp12==0)],
            ResJac$score[which(rowsumhelp12==1)])
Kind <- c(rep("one method",
              length(ResJac$score[which(rowsumhelp12==0)])),
          rep("both methods",
              length(ResJac$score[which(rowsumhelp12==1)])))
scorec <- data.frame(ScoreC, Kind)

pdf("images/score_compare.pdf")
boxplot(scorec$ScoreC~scorec$Kind, ylab="Score")
dev.off()


# conclusions on the right release level
# with results obtained by the Jaccard Index
##################################################

# number of observed biclusters for each run
nb <- matrix(nrow=11,ncol=100)
nbv <- c()
for (i in 1:11) {
  for (j in 1:100) {
    nbv <- c(nbv, RES[[i]][[j]]@Number)
  }
}
# vector of release level belonging to each result
```

```
thr <- sort(rep(seq(0.51,0.71,0.02), 100))
thr.total <- rep(thr, nbv)

# observed bicluster vs. release level
pdf("images/release2.pdf")
boxplot(nbv~thr, xlab="Release Level",
        ylab="Obtained bicluster")
dev.off()

# run number of resulting bicluster
br <- rep(1:1100,nbv)
nr <- br[ResJac$clustindtotal]

# bicluster number of resulting bicluster
ind2 <- which(ResJac$clustindtotal==TRUE)

# amount of resulting biclusters per release
table(thr[nr])
pdf("images/release.pdf")
barplot(table(thr[nr]), ylim=c(0,10),
        xlab="Release level", ylab="Frequency")
dev.off()

# score vs. release level
scoreJac <- ResJac$score
pdf("images/score_release.pdf")
plot(scoreJac~thr.total[ind2], xlab="Release level",
     ylab="Score")
dev.off()

# bicluster-size vs. release level
gr <- rsize*csize
m051 <- mean(gr[thr.total==0.51])
m053 <- mean(gr[thr.total==0.53])
m055 <- mean(gr[thr.total==0.55])
m057 <- mean(gr[thr.total==0.57])
m059 <- mean(gr[thr.total==0.59])
m061 <- mean(gr[thr.total==0.61])
m063 <- mean(gr[thr.total==0.63])
m065 <- mean(gr[thr.total==0.65])
m067 <- mean(gr[thr.total==0.67])
m069 <- mean(gr[thr.total==0.69])
m071 <- mean(gr[thr.total==0.71])
par(mfrow=c(1,3))
```

```
pdf("images/size.pdf")
boxplot(rsize~thr.total,xlab="Release level",
        ylab="row size")
boxplot(csize~thr.total,xlab="Release level",
        ylab="column size")
boxplot(gr~thr.total,xlab="Release level",
        ylab="total size")
points(1:11, c(m051, m053, m055, m057, m059, m061,
       m063, m065, m067, m069, m071), type="b", col="red")
dev.off()

# similar biclusters between each release level
JAC01 <- ClustJac[[5]]>0.9
similar.thr <- c()

for (y in seq(0.51,0.71,0.02)) {
  for (x in seq(0.51,0.71,0.02)) {
    yhelpindex <- thr.total==y
    xhelpindex <- thr.total==x
    similar.thr <- c(similar.thr, sum(
                    c(JAC01[yhelpindex, xhelpindex])))
    }
  }
similar_total <- matrix(similar.thr, ncol=11)
similar_total

# separate for rows and columns
Jac_sep <-jaccard.sep(rowsnew, colsnew)
jacr <- Jac_sep[[1]]
jacc <- Jac_sep[[2]]
lower <- lower.tri(jacr, diag=FALSE)
jacr[lower] <- t(jacr)[lower]
diag(jacr) <- 0
lower <- lower.tri(jacc, diag=FALSE)
jacc[lower] <- t(jacc)[lower]
diag(jacc) <- 0
save(jacr, jacc, file="RData/JacSeparate.RData")

jacr01 <- jacr>0.9
jacc01 <- jacc>0.9
similar.thr <- c()

for (y in seq(0.51,0.71,0.02)) {
  for (x in seq(0.51,0.71,0.02)) {
    yhelpindex <- thr.total==y
    xhelpindex <- thr.total==x
    similar.thr <- c(similar.thr, sum(
                        c(jacr01[yhelpindex, xhelpindex])))
    }
  }
similar_sepr <- matrix(similar.thr, ncol=11)

similar.thr <- c()

for (y in seq(0.51,0.71,0.02)) {
  for (x in seq(0.51,0.71,0.02)) {
    yhelpindex <- thr.total==y
    xhelpindex <- thr.total==x
    similar.thr <- c(similar.thr, sum(
                        c(jacc01[yhelpindex, xhelpindex])))
    }
  }
similar_sepc <- matrix(similar.thr, ncol=11)

similar_sepr
similar_sepc

# adjusted jaccard index on results
# with an level of 0.59, 0.61, 0.69
subrows1 <- rowsnew[,thr.total==0.59]
subrows2 <- rowsnew[,thr.total==0.61]
subrows3 <- rowsnew[,thr.total==0.69]
subrows <- cbind(subrows1, subrows2, subrows3)
subcols1 <- colsnew[,thr.total==0.59]
subcols2 <- colsnew[,thr.total==0.61]
subcols3 <- colsnew[,thr.total==0.69]
subcols <- cbind(subcols1, subcols2, subcols3)

Jac_adj <- jaccard.adj(subrows, subcols)
save(Jac_adj, file="RData/JacAdj.RData")

Jac_adj1 <- Jac_adj==1

thr.new <- rep(c(0.59, 0.61, 0.69), c(sum(thr.total==0.59),
               sum(thr.total==0.61), sum(thr.total==0.69)))
similar.thr <- c()

for (y in c(0.59, 0.61, 0.69)) {
  for (x in c(0.59, 0.61, 0.69)) {
    yhelpindex <- thr.new==y
    xhelpindex <- thr.new==x
    similar.thr <- c(similar.thr, sum(c(
                    Jac_adj1[yhelpindex, xhelpindex])))
    }
  }

similar_adj <- matrix(similar.thr, ncol=3)
similar_adj
```

E. CD-ROM

The attached CD-ROM contains the whole R-Code and Bash-Script described above, as well as the TCGA data set, the resulting .RData files, the generated graphics and a digital version of the thesis in hand. An small overview over the content of the included folders is given below:

- **biclust:** R files and bash scripts to run the plaid model for all release levels.
- **data:** TCGA data set in .txt format.
- **images:** All generated graphics in .pdf format.
- **RData:** R workspace files in .RData format.
  - R files in .r format.
  - Readme file in .txt format.
  - This thesis in .pdf format.

Affidavit

I, Georg Pfundstein, hereby declare that this bachelor-thesis in question was written single-handed and no further as the denounced resources and sources were employed.

Munich, 8th February 2010

Georg Pfundstein

Eidesstattliche Erklärung

Hiermit erkläre ich, Georg Pfundstein, dass es sich bei der vorliegenden Bachelorarbeit um eine selbständig verfasste Arbeit handelt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden.

München, den 8. Februar 2010

Georg Pfundstein